Architecture-Centric Support for Security Orchestration and Automation

Chadni Islam

School of Computer Science

The University of Adelaide

This dissertation is submitted for the degree of

Doctor of Philosophy

Supervisors: Professor Muhammad Ali Babar and Dr Surya Nepal

November 2020

Table of Contents

List of Figures

List of Tables

Abstract

Security Orchestration, Automation and Response (SOAR) platforms leverage integration and orchestration technologies to (i) automate manual and repetitive labor-intensive tasks, (ii) provide a single panel of control to manage various types of security tools (e.g., intrusion detection system, antivirus and firewall) and (iii) streamline complex Incident Response Process (IRP) responses. SOAR platforms increase the operational efficiency of overwhelmed security teams in a Security Operation Centre (SOC) and accelerate the SOC's defense and response capacity against ever-growing security incidents. Security tools, IRPs and security requirements form the underlying execution environment of SOAR platforms, which are changing rapidly due to the dynamic nature of security threats. A SOAR platform is expected to adapt continuously to these dynamic changes. Flexible integration, interpretation and interoperability of security tools are essential to ease the adaptation of a SOAR platform. However, most of the effort for designing and developing existing SOAR platforms are ad-hoc in nature, which introduces several engineering challenges and research challenges. For instance, the advancement of a SOAR platform increases its architectural complexity and makes the operation of such platforms difficult for end-users. These challenges come from a lack of a comprehensive view, design space and architectural support for SOAR platforms.

This thesis aims to contribute to the growing realization that it is necessary to advance SOAR platforms by designing, implementing and evaluating architecture-centric support to address several of the existing challenges. The envisioned research and development activities require the identification of current practices and challenges of SOAR platforms; hence, a Multivocal Literature Review (MLR) has been designed, conducted and reported. The MLR identifies the functional and non-functional requirements, components and practices of a security orchestration domain, along with the open issues. This thesis advances the domain of a SOAR platform by providing a layered architecture, which considers the key functional and non-functional requirements of a SOAR platform. The proposed architecture is evaluated experimentally with a Proof of Concept (PoC)

system, Security Tool Unifier (STUn), using seven security tools, a set of IRPs and playbooks. The research further identifies the need for and design of (i) an Artificial Intelligence (AI) based integration framework to interpret the activities of security tools and enable interoperability automatically, (ii) a semantic-based automated integration process to integrate security tools and (iii) AI-enabled design and generation of a declarative API from user query, namely DecOr, to hide the internal complexity of a SOAR platform from end-users. The experimental evaluation of the proposed approaches demonstrates that (i) consideration of architectural design decisions supports the development of an easy to interact with, modify and update SOAR platform, (ii) an AI-based integration framework and automated integration process provides effective and efficient integration and interpretation of security tools and IRPs and (iii) DecOr increases the usability and flexibility of a SOAR platform. This thesis is a useful resource and guideline for both practitioners and researchers who are working in the security orchestration domain. It provides an insight into how an architecture-centric approach, with incorporation of AI technologies, reduces the operational complexity of SOAR platforms.

Declaration

I, Chadni Islam, certify, this work contains no material which has been accepted for the award of any other degree or diploma in my name in any university or other tertiary institution. In addition, I certify that no part of this work will, in the future, be used in a submission in my name for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint award of this degree. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

I acknowledge that the copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

I acknowledge the support I received for my research through the provision of Data61 postgraduate scholarship and full-fee scholarship.

Signature: ____ _

Date: 31.07.2020

Acknowledgments

This thesis would not be possible without the continuous guidance, encouragement and support of several persons. I would like to take this opportunity to acknowledge them.

First, I would like to express my gratitude and sincere appreciation to my supervisor, Professor M. Ali Babar for his outstanding support, encouragement, and unwavering guidance during all stages of my PhD. I am deeply indebted and grateful for all the discussions and brainstorming sessions that we had through this timeframe. He was instrumental in defining the path of my research. He continuously guided me and encouraged me to be professional and do the right thing even when the road got tough. Without his persistent help, the goal of this thesis would not have been realized. He helped me to understand the power of critical thinking and reasoning. He supported me by providing numerous revisions on my writing and comments on papers. He also gave me opportunities to be involved in student projects and engaged with industry partners and other researchers. I am greatly fortunate to get the chance to work closely with and mentored by him. His passion and scientific intuition profoundly motivated me towards high-quality research and inspired me to continually excel beyond ordinary limits and kept me focused on my research.

I would also like to extend my deepest gratitude to my co-supervisor Dr. Surya Nepal. His constructive criticism helped me to develop a broader perspective in my research. I am highly fortunate to be mentored by him. The fruitful discussion with Surya helped me to bring the thesis in shape. The completion of my thesis would be possible without the support and nurturing of his supervision. His encouragement to participate in symposium and present research to a wider community helped me to get feedback from different people and enrich the contribution. I would like to recognize the invaluable assistance that he provided during my study.

I would like to pay my special regards to CREST team members. All their efforts are greatly appreciated. Special thanks to Dr Faheem Ullah, Triet Mihn Le, Dr Mojtaba Shahin, Dr Aufeef Chauhan, Dr Nguyen Khoi Tran and Bushra Sabir. Faheem and Triet had helped by continuously reviewing my research paper and providing constructive

criticism. Faheem and Aufeef's expertise in architecture and Triet and Bushra's expertise in machine learning facilitated my learning and enriching the work. They were always open to discussion. The fruitful discussion with them helped me to look at my research from different perspectives. I would also like to acknowledge, Dr. Mansooreh Zahedi for the encouragement and insightful discussions in many stages of my candidature.

I also acknowledge CSIRO's Data61, School of Computer Science and CREST for providing various scholarships to facilitate and pursue my doctoral research and activities.

I must acknowledge my best friend and partner, Sheik Mohammad Mostakim Fattah, without whose support, I would not have finished this thesis, who has constantly encouraged me through the hardest time of my PhD. I would also like to thank my Mom, Sister, Papa, Mamoni and Brother for their encouragement. I thank all my friends and family members who directly or indirectly supported me throughout this time.

Chapter 1

Introduction

The rapid growth in cyberattacks has recently become a major concern for industry and governments. Advanced data exploitation and phishing techniques are used to attack most organizations [1-3]. Industries ranging from financial to healthcare are all vulnerable to cyberattacks; among them, healthcare, manufacturing, financial services, government agencies and educational institutes are the most targeted [1, 4-7]. According to Cybersecurity Ventures, global cybercrime damage may reach $6 trillion (£4.6 trillion) in 2021 [8, 9]. In Australia, all levels of government, industry, political organizations, education, health, essential services providers and operators of critical infrastructure are experiencing a significant increase in cybercriminal activities and sophisticated state-based cyberattacks [10-12]. This was been announced publicly by the Australian Prime Minister in June 2020 to raise awareness [11, 12].

Many organizations are building Security Operation Centers (SOC) to improve their organization's security position, continuously monitor, detect, prevent, analyze and respond to, and reduce the financial impact of cyberattacks [13-15]. According to the SANS (Escal Institute of Advanced Technologies) Institute, "*SOC is a combination of people, processes and technology protecting the information system of an organization through projective design and configuration, ongoing monitoring of the system state, detection of unintended actions or undesirable states and minimizing damage of unwanted effects*" [13]. SOCs help organizations to increase their ability to detect threats and respond faster to cyberattacks; hence, minimizing the damage of cyberattacks. A recent survey by Kaspersky has revealed that the financial damage to many organizations is reduced to half with the help of SOCs [14]. An organization's SOC uses a variety of security tools, developed by different vendors, to protect its Information Technology (IT)

infrastructures and Business Applications (BA). Security software or Security tools are software-intensive systems for detecting, preventing and recovering from cyberattacks. In this thesis, we have used the term *security tool* to refer to security software/security tools and information systems that are used or participate in responding to security incidents. Some of the commonly used security tools are antivirus, firewall, Intrusion Detection Systems and Instruction Prevention Systems (IDS/IPS), Security Information and Events Management (SIEM) and Endpoint Detection and Response (EDR) tools [16-19]. These security tools use different steps for threat defense, from detecting, preventing and responding to security incidents to performing data enrichment by correlating and analyzing event data with contextual information.

Security teams of a SOC are expected to monitor and analyze the activities (e.g., validate alerts, correlate logs, remove malware, etc.) of these security tools to respond to security incidents [20-23]. We refer to the human experts who are involved in different activities of SOC to protect an organization from cyberattacks as security teams. Common roles of security teams are: Cyber Security Incident Response Team (CSIRT), threat hunter, forensic analyst, security analyst, security administrator and network administrator [17, 33, 34]. To increase operational efficiency, SOCs are adopting Security Orchestration, Automation and Response platforms (SOAR) that orchestrate the activities of security tools and human experts and automate the labor-intensive repetitive tasks that are performed manually by security teams [20, 23, 24]. In this thesis, the terms *security orchestration and automation*, *security orchestration* and *SOAR* are used interchangeably.

Security orchestration is defined as "*the planning, integration, cooperation and coordination of the activities of security tools and experts to produce and automate required actions in response to any security incident across multiple technology paradigms*" [22]. The incorporation of security orchestration and automation technologies promises to solve several challenges faced by the overwhelmed security teams in a SOC dealing with complex security operations. A key challenge is to analyze a huge pool of alerts in time.

Incident responders must immediately respond to contain, mitigate and minimize the damage of a security incident. Another challenge is that most of the Incident

Response Processes (IRP) contain a sequence of activities that are required to be performed manually by a security team. Performing a task manually is often error-prone and time-consuming. In addition, most organizations have a lack of security experts to operate their SOCs efficiently. For instance, a SANS survey has revealed that a SOC's capabilities may be hampered due to the lack of skilled professionals [13]. A SOAR platform minimizes the burden on human experts by orchestrating, automating and streamlining IRPs. It is a must-have technology for organizations (in-house or through service providers) to address the challenges associated with a massive volume of security alerts, dynamic security threat landscapes and huge skill gaps [2, 25, 26].

The adoption of SOAR platforms has increased significantly in the last couple of years [26, 27]. Some of the examples of SOAR platform security vendors include (but are not limited to): MacAfee [28], IBM [22], FireEye [29], Intel [30], AlienVault [31], Swimlane [32], LogRhythm [33] and Demisto [34]. These vendors report on successful adoption of their SOAR technologies in SOC environments and how SOAR platforms may help to increase the operational efficiency of security teams.

Figure 1.1 depicts an overview of a SOAR platform. To develop or deploy a SOAR platform, SOAR designers or developers first *assess* an organization's existing security tools, information systems, security requirements and security expertise. Based on the assessments, they develop or utilize existing APIs, plugins or scripts to *integrate security tools* into a SOAR platform. Integration of security tools allows security teams to access the security tools from a single platform. Furthermore, *response processes/ IRPs, rules and playbooks* (i.e., an automated workflow) are *designed* to orchestrate and automate IRPs in response to security incidents [20, 32, 35]. Playbooks are mostly built from IRPs by demystifying security requirements/ concerns [26]. They help to deploy countermeasures following day to day security practices.

The integrated security tools generate alerts, logs and reports in different formats that are used by a wide range of security teams. Data generated by different security tools are gathered and stored for investigation by security teams. Different security teams work with a different set of security tools to respond to security incidents. Furthermore, the data generated by one security tool might be used by other security tools for further processing. For example, the alerts produced by different IDS are analyzed by a SIEM

tool for a gathering context. As security tools have different data formats and are mostly designed to work in isolation, a SOAR platform needs to work as an interpreter among the security tools and enable interoperability among them. Rules, mechanisms or

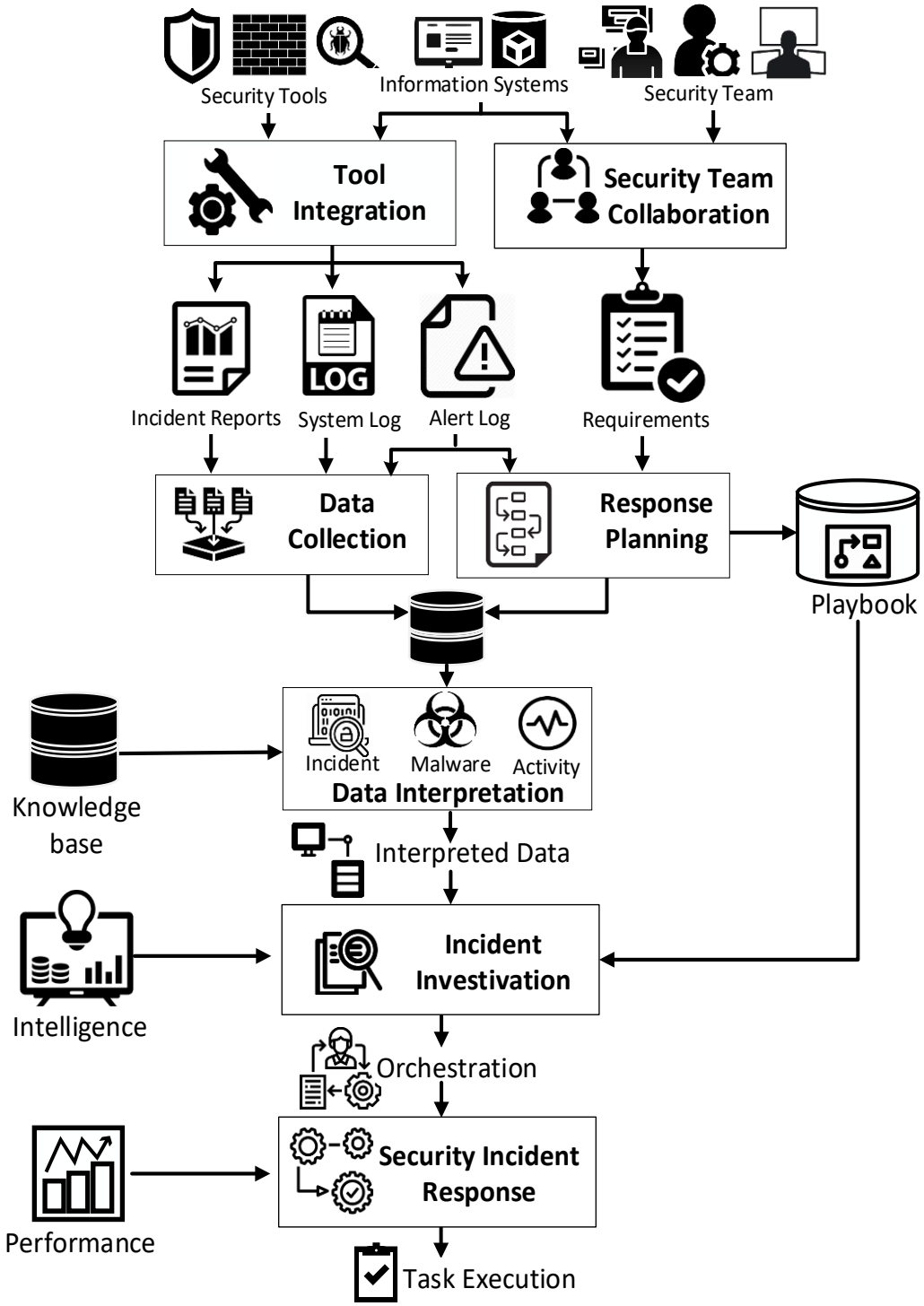


Figure 1.1 An overview of a typical security orchestration and automation platform

knowledge bases are developed for *interpretation of the gathered data* by a SOAR platform to bring them to security teams for analysis and investigation. Security teams *investigate these data* utilizing security intelligence based on defined playbooks. Finally, responses towards a security incident are orchestrated, and tasks are executed by a SOAR platform invoking the integrated security tools.

1.1 OBJECTIVES AND RESEARCH QUESTIONS

The rapid proliferation of cyberattacks is changing the way a SOC defends and responds to security incidents. Due to the COVID-19 pandemic, organizations are under a new level of stress where most people are working from home, resulting in COVID-19 themed cyberattacks that are phishing, business email compromise scams, ransomware, remote working vulnerabilities, hacking and hijacking of video and teleconferences [36-39]. According to the Australian Cyber Security Centre (ACSC), the frequency and severity of COVID-19-themed cyberattacks are likely to increase in the coming days [40]. In recent years, SOCs have evolved significantly to encompass a sophisticated range of security tools and activities within organizations to build and operate their cybersecurity tools. According to Cybersecurity Ventures, the global spending on cybersecurity products and services (i.e., security tools) will exceed $1 trillion cumulatively over five years (i.e., from 2017 to 2021) [8, 9]. The Australian federal government has announced a budget of $1.35 billion for enhancing Australian cybersecurity capabilities over the next ten years. According to AustCyber (i.e., Australian Cyber Security Growth Network), by 2026, approximately 77% of cybersecurity expenditure will be spent on externally managed security services [41].

To counter the emerging security threats and make use of increasing demand for the available security products and services, new security tools are being introduced to secure organizational IT infrastructures, which are increasingly becoming hyper-virtual, mobile and connected. This situation causes a continuous change in the underlying execution environment (e.g., security tools, integration mechanisms and security requirements) of a SOAR platform, which must be designed in such a way that it can evolve with the evolution of its execution environment.

Software architecture plays an integral role in the design of large-scale integrated systems [42-44], like SOAR platforms. It guides the design, development and evaluation of a software system over time. Bass et al. have defined software architecture as "*a set of structures needed to reason about the system, which comprises software elements, relations among the elements and properties of both the elements and the relations*" [45]. Software architecture abstracts the different elements of a system and the relationships among these elements with respect to the functional and non-functional requirements (also known as quality requirements) of a system [43]. Software architecture is also known as the composition of architectural design decisions [44]. Jansen et al. have defined architectural design decisions as "*a description of the set of architectural additions, subtractions and modifications to the software architecture, the rationale and the design rules, design constraints and additional requirements that (partially) realize one or more requirements on a given architecture*". However, the existing SOAR platforms lack the proper abstraction for designing a SOAR platform at an architectural level. The research reported in this thesis has been motivated by the increased realization that there is an important and urgent need for architecture-centric support for designing and evolving SOAR platforms, which are expected to integrate easily and smoothly and be interoperable with existing and new security tools.

Before delving into the main body of this thesis, we perform an extensive study of the state-of-the-art and the state-of-the-practice of the existing SOAR platforms by carrying out a Multivocal Literature Review (MLR) [46]. More specifically, we investigate (i) the key functionalities and components required of a SOAR platform (ii) drivers of a SOAR platform and (iii) variation in SOAR platform solutions. Through an analysis of the existing studies, we find that there is also a lack of common understanding among vendors and SOCs of SOAR platforms. Some vendors are simply providing automated workflows or playbooks and claiming that as a SOAR platform. Some are providing an integrated layer for security tools and claiming them to be a SOAR platform, whilst others confuse security orchestration with security automation. Most of the existing SOAR platforms have been implemented in an ad-hoc manner, based on organizational requirements, without much attention paid to the underlying infrastructure. The lack of comprehensive vision and ad-hoc design results in several challenges in evolving a SOAR platform over time.

Some of the key challenges are: (i) seamless integration of new security tools and new playbooks, (ii) managing interoperability among the isolated and heterogenous security tools in a changing environment and (iii) building capability in a security team to understand the underlying libraries and components of a SOAR platform to incorporate new tools. To bridge the gap, we design, implement and evaluate architecture support for a human-centric SOAR platform. Based on the proposed architecture, we propose a set of Artificial Intelligence (AI) enabled toolsets and frameworks that address the abovementioned challenges.

Thesis statement: A SOAR platform is an integrated platform that involves the realization of three paradigms – unification, orchestration and automation. Integration mechanisms and interactions of security teams with a SOAR platform influence the usefulness and large-scale realization of existing SOAR platforms. An abstraction of a SOAR platform, along with its key functionalities, is required to identify the suitable architecture style and architecture patterns to embed agility in a human-centric SOAR platform.

This thesis aims to propose architecture-centric support for integrating security tools into a SOAR platform, where a SOAR platform works as a hub for security tools and security teams. To achieve the goal, we address three key Research Questions (RQ).

RQ1. How has security orchestration been defined and what are the key challenges in security orchestration?

Since security orchestration and automation is an emerging concept, there is a lack of consensus amongst vendors about the various functionalities, components, toolsets and challenges of security orchestration. To identify how security orchestration has been defined, we investigate "*what is security orchestration?*", "*what are the key functional and Non-Functional Requirements (NFR) of a SOAR platform*" and "*what types of solutions has been presented?*". To the best of our knowledge, there has been no effort to systematically review and analyze the existing SOAR platforms' functional features, NFR and core components for designing a human-centric SOAR platform. Therefore, it is important to systematically review state-of-the-art of security orchestration and automation to (i) identify the key functional and NFR requirements,

(ii) gain insight about the key components and technologies that can fulfill the essential functional and NFR requirements and (iii) identify and codify the challenges associated with current practice for designing and deploying security orchestration and automation platforms.

RQ2. How does software architecture play a role in improving the design practice of security orchestration and automation platforms?

The functional and NFR requirements used to design and deploy a SOAR platform depend greatly on the requirements of security teams and the underlying infrastructure of an organization. As different security teams have different requirements, ad-hoc approaches appear to be more dominant in designing and deploying a SOAR platform. Incorporation of the changes in the underlying execution environment without a clear guideline and view of a SOAR design space results in a monolithic and complex design that is hard to evolve. Lack of conceptual and practical guidelines for optimal architectural design decisions may result in a highly complex design. It can be argued that an architecture-centric approach can help to reduce the design complexity of a SOAR by modularizing the functional and NFR elements, alongside consideration of architectural design decision help with analyzing and understanding sub-optimal design decisions that can be improved by leveraging well-known architecture styles and patterns.

RQ3. What kinds of tools and techniques can be incorporated to realize the architecture while fulfilling the functional and NFR of the implemented platform?

There is a need to identify the tools and techniques that are suitable for the realization of the proposed architecture, to provide an effective and efficient way to adapt to the changes in a SOAR platform. We designed the following three (sub) research questions to answer RQ3.

RQ3.1. How is it possible to enable seamless interoperability and interpretability among security tools and SOAR platforms?

A SOAR platform needs to enable interoperability among different security tools to orchestrate and automate IRPs. To do this, it is necessary to interpret both the capabilities of security tools and activities of IRPs, and map which security tools to use

to perform which activities. The whole process is done manually by a security team. Thus, changes in any of these components require security staff to adapt the changes manually. As new security tools and IRPs evolve with emerging threats, a SOAR platform needs to support easy integration of new security tools and IRPs. An integration framework can be designed leveraging Artificial Intelligence (AI) technologies such as semantic technologies, Natural Language Processing (NLP) and Machine Learning (ML) tools and techniques. Semantic-based integration of security tools can provide the security team with the flexibility to integrate security tools easily, without worrying about the underlying integration mechanisms. In addition, NLP and ML-based interpretation and prediction of IRPs reduces the burden on the security team to map the IRPs with the activities of the security tools.

RQ3.2. How is it possible to automate the process of integrating security tools in a SOAR platform?

To integrate a security tool in a SOAR platform, a security expert needs to know the underlying libraries and integration mechanism (e.g., APIs, plugins and scripts) of both security tools and SOAR platforms. Integration of security tools is considered one of the key challenges of a SOC as security tools vary in terms of their type (e.g., proprietary, legacy and open source), and the structure of the generated and consumed data. The process of integration is also repetitive and manual, whereby a security team first needs to investigate the type of security tool, its capabilities, the activities it can execute, and map the activity of IRPs with the security tools. The process of integration of security tools in a SOAR can be automated by designing a semantic-based integration approach.

RQ3.3. How is it possible to hide the internal complex architecture of a SOAR platform from the security team?

Incorporation of different automation, orchestration and AI technologies in a SOAR platform results in a complex architecture that makes the operating of such a platform difficult for an end-user with a changing threat landscape. Thus, there is a need to hide the underlying complex design of a SOAR platform from its end-users to reduce the operating complexity. A set of declarative APIs can be designed by leveraging AI technologies such as NLP and semantic tools and techniques. Declarative APIs are a

form of API through which a security team can provide the command in declarative form to specify what a SOAR platform needs to do without specifying the details as to how to execute that command.

1.2 THESIS OVERVIEW

This section presents a summary of how we address each of the RQs, which are analyzed across five chapters. Figure 1.2 provides an overview of the thesis. We describe the focus of each chapter below.

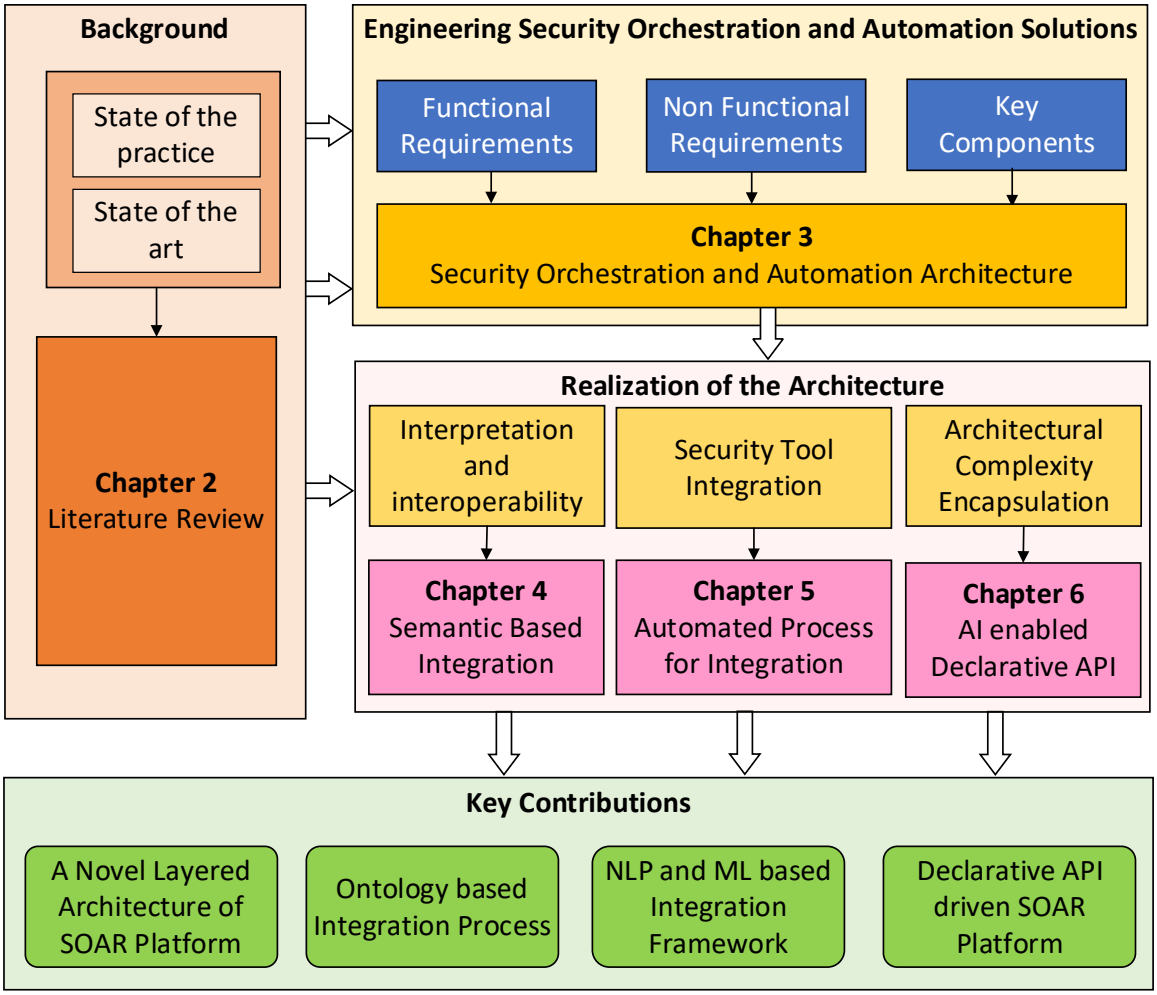


Figure 1.2 Overview and scope of the thesis

Chapter 2 Literature Review

We performed a comprehensive study of the existing security orchestration solutions, tools and technologies to answer RQ1 (orange boxes of Figure 1.2). A Multivocal Literature Review (MLR) has been conducted for this and is reported in

Chapter 2. Chapter 2 demonstrates a thorough knowledge of the area and provides an argument to support the thesis focus. More specifically, it highlights the importance of having an architecture for a SOAR platform. The review has helped us to gain a comprehensive understanding of the security orchestration domain. The review has identified the functional and Non-Functional Requirements (NFR) of a SOAR platform, along with the core components and technologies required to provide the functional requirement and fulfill the NFRs (blue boxes). It works as a guideline for any researcher or practitioner who plans to deploy orchestration and automation technologies in their SOC.

Chapter 3 Security Orchestration and Automation Architecture

One of the key purposes of a SOAR platform is to address the challenges of a SOC with integrating security tools and operation activities. To design a SOAR platform to fulfil this purpose, an architecture is proposed for a new kind of SOAR platform (yellow boxes of Figure 1.2). This chapter addresses RQ2 by presenting a concept map and key dimensions of the architecture design space for integration of security tools and operational activities in a SOAR platform. The architecture is designed considering the key functional and NFR of a SOAR platform. This chapter proposes a high-level architecture for a SOAR platform that relies on the layered architectural style. The proposed layered architecture consists of six layers – a security tool layer, integration layer, semantic layer, data processing layer, orchestration layer and User Interface (UI) layer. An abstraction layer is considered as part of the UI layers. By providing a detailed description of each layer of the proposed architecture, the way the layers integrate to achieve a set of NFRs, including *integrability*, *interpretability*, *interoperability*, *modifiability* and *usability*, can be shown. The proposed architecture is evaluated based on a Proof of Concept implementation of a SOAR platform for two use case scenarios with seven security tools. The realization of the architecture for different purposes is presented in Chapters 4, 5 and 6 (purple boxes of Figure 1.2).

Chapter 4 Semantic-Based Integration

This chapter answers RQ3.1 by addressing the challenges associated with changes in the underlying execution environment of a SOAR platform that may hamper the interpretability and interoperability of security tools for automated execution of an

IRP. An integration framework is proposed to unify the security tools' data by formalizing the security tools' capabilities, inputs, outputs and the activities of IRPs. An ontological knowledge base is developed that formalizes the security tools' capabilities and activities and defines their relationship. The integration framework contains an interoperability model to enable interoperability amongst the security tools for the automated execution of a sequence of activities. Interpreting the activities and capabilities to execute these activities, it finds the appropriate security tools that can be used for their execution. It shows how NLP and ML techniques can be used along with semantic technology to automate the interpretation of activities. A learning-based approach is proposed to identify activity classes from new activities' descriptions given in natural language. The proposed approach is evaluated based on seven security tools and 23 IRPs.

Chapter 5 Automated Process for Integration

This chapter answers RQ3.2 by addressing the challenges with manual design and development of integration technologies (i.e., APIs. Plugins and scripts) for security tool integration. It leverages the semantics technologies used for formalizing the key concepts of security tools and IRPs to automate the process of integrating security tools in a SOAR platform. The integration process automates the selection of security tools, interpretation of security tools' capabilities, formulation of commands to invoke security tools and finally invocation of security tools to execute an activity. This chapter mainly realizes three layers – the data processing, semantic and security tool layers. The data processing layer deals with information related to security tools. The semantic layer provides information related to the semantics of input and output, as well as the activities that are executed by security tools and directly related to the orchestration layer. The orchestration layer activates the tasks that are required when integrating multiple security tools. The proposed approach is evaluated based on an experiment using IRP for Distributed Denial of Service (DDoS) attacks.

Chapter 6 AI Enabled Declarative API

The proposed layered architecture has an abstraction layer that plays a key role in hiding the internal complex architecture of a SOAR from its end users (i.e., the security team). We find that most of the abstractions through declaration are more prominent and

gain major attention from recent software developments. As a result, to answer RQ3.3., this chapter proposes a set of declarative APIs for a SOAR platform. The declarative API can easily be used by the developers or end-users to define their plans, integrate security tools, update IRPs or even update the knowledge base. AI based approaches such as semantic and NLP technologies are leveraged in the design of the declarative APIs that allow users with little knowledge about declarative APIs to provide the command through natural language. A semantic framework is proposed to automate the generation of the declarative APIs from the task description.

1.3 THESIS CONTRIBUTIONS

This section summarizes the key contributions of this thesis that have been made while answering the Research Questions (RQs) (green boxes of Figure 1.2). We identify the key functional and NFR requirement of a human-centric SOAR platform (i.e., answer RQ1). Considering the functional and NFR requirements, we propose a layered architecture for integrating security tools in a SOAR platform (i.e., answer RQ2). We further leverage AI technologies to implement and evaluate a proof of concept SOAR platform (i.e., answer RQ3). The key contributions of this thesis are summarized as follows:

- We establish a solid background knowledge of security orchestration and automation research and practices. We identify the key challenges that practitioners and researchers are expected to overcome through security orchestration. We also provide a taxonomy of different aspects of security orchestration practices that includes *the key functional and NFR requirements* and automation strategies. Furthermore, the open issues of the existing SOAR platforms are presented from people, process and technology perspectives. These contributions answer RQ1, are presented in Chapter 2 and are published in the ACM Computing Survey (Impact factor: 6.131, Core 2020 Ranking **A***) as:
    - Chadni Islam, Muhammad Ali Babar, and Surya Nepal. 2019. A Multi-Vocal Review of Security Orchestration. *ACM Computing Survey*. (CSUR) Vol 52, Issue 2, Article 37 (April 2019), 45 pages.

- We demonstrate that the successful realization and evolution of a SOAR platform are governed by how the security tools are integrated, orchestration processes are defined, and security teams communicate with such a platform. We propose a layered architecture for a SOAR platform, which forms the basis for enabling automated integration of security tools and automated interpretation of the activities performed by a SOAR platform. We demonstrate that one of the basic requirements for any large-scale system is to have an abstract layer that hides the internal complexity of a platform from its end-user. Hence, we propose a layered architecture with an abstraction layer to fulfill this requirement. We further provide a proof of concept SOAR that is designed and implemented based on the proposed architectural approach. These contributions answer RQ2, are presented in Chapter 3 and were accepted for publication by the 14th European Conference on Software Architecture (ECSA'2020) (Core 2020 Ranking **A**) as:
    - Chadni Islam, Muhammad Ali Babar, and Surya Nepal. Architecture-centric Support for Integrating Security Tool in a Security Orchestration Platform. *14th European Conference on Software Architecture (ECSA'2020)*, 14-18 September 2020, L'Aquila, Italy.
- We propose a semantic-based integration framework for automated interpretation of security tools and activities of IRPs to enable interoperability among security tools and automate the execution of IRPs. The integration framework consists of an ontological model, a prediction module and an annotation module. Considering a SOAR platform cannot automatically interpret activities of IRPs, security tool capabilities, and their input and generated data, we formalize various inputs, outputs and capabilities of security tools, activities of the IRPs and mapping of activities with the security tools' capabilities in an ontological knowledge base. A systematic and structured path is followed to define and annotate classes of the ontology. The prediction module is developed by utilizing NLP and ML techniques that (i) learn the semantic model and (ii) automatically categorize the activities of IRPs according to the activity classes of an ontology. These contributions answer RQ3.1, and are presented in Chapter 4 and were published by the 31st International Conference on

Advanced Information Systems Engineering (CAiSE'2019) (CORE 2018 Ranking A) as

  - Chadni Islam, Muhammad Ali Babar and Surya Nepal. Automated Interpretation and Integration of Security Tools Using Semantic Knowledge. *In 31st International Conference on Advanced Information Systems Engineering (CAiSE'2019)*, June 3-7, 2019, Rome, Italy.

- We propose an ontology-driven approach for automating the process of integrating security tools in a SOAR platform. Following this process, data generated and consumed by security tools are interpreted and integrated automatically. We have identified that each security tool has a set of capabilities where the orchestration process has a set of activities. To respond to an activity, one security tool might require the output of other security tools. The proposed integration process formulates the input of security tools by deconstructing and extracting the features from the output of other security tools where needed. These contributions answer RQ3.2, and are presented in chapter 5 and were published by the 2019 International Conference on Software and Systems Process (ICSSP'2019) (CORE 2020 Ranking A) as

  - Chadni Islam, Muhammad Ali Babar and Surya Nepal. An Ontology-Driven Approach to Automating the Process of Integrating Security Software Systems. *In ICSSP 2019 International Conference on Software and Systems Process (ICSSP'2019)*, May 25-26, 2019, Montreal, Canada.

- We demonstrate that the incorporation of different technologies increases the architectural complexity of a SOAR platform. Different teams have different requirements or use a SOAR platform for different purposes. We identify the requirements of a set of declarative APIs through which security teams can provide their task without having a detailed knowledge of the underlying infrastructure of a SOAR platform. To free security teams from learning the declarative API, we propose an AI-based approach for generating declarative APIs from task description. This way novice users or users with little knowledge about the declarative API would be able to provide the commands to execute a task or activity in a SOAR platform. These contributions answer RQ3.3, are presented in Chapter 6 and were submitted to

the journal of Transactions of Software Engineering and Methodology (TOSEM) (Impact Factor: 2.071, Core 2020 Ranking **A***) as

- Chadni Islam, Muhammad Ali Babar and Surya Nepal. AI-Enabled Design and Generation of Declarative API for Security Orchestration Platform. ***Submitted*** to *Transactions of Software Engineering and Methodology (TOSEM)*.

Each chapter of this thesis is focused on addressing a research question while contributing to the overall objective of the thesis. We have mentioned the needed additional functionality or integration in each chapter separately. The limitations of the proposed approaches have also been discussed at each chapter.

1.4 THESIS ORGANIZATION

The remainder of this thesis is organized as follows. Chapter 2 presents a comprehensive review of the security orchestration domain. Chapter 3 provides a novel architecture for supporting the integration of security tools and operation activities in a human-centric SOAR platform. Chapter 4 presents an integration framework to semantically integrate security tools in a SOAR platform and provides an automated approach for interpreting security tools and activities of IRPs. Chapter 5 presents a process for automatically integrating security tools in a SOAR platform based on an ontological knowledge base. Chapter 6 presents a set of declarative APIs to hide the inherent complexity of a SOAR platform and an AI-based approach to automate the generation of declarative APIs from a user query. Chapter 7 concludes the thesis by providing a road map of two significant avenues for future work.

Chapter 2

Literature Review

A Security Orchestration, Automation and Response (SOAR) platform aims to integrate multivendor security tools, so that the security tools can effectively and efficiently interoperate to automate and streamline activities of security teams in a Security Operation Centre (SOC). Given the growing need and importance of SOAR platforms, there has been an increasing amount of literature on their different aspects. However, there has been no effort to systematically review and analyze the reported solutions. This chapter aims to identify different aspects of SOAR platforms. To realize this goal, a Multivocal Literature Review (MLR) has been carried out, which systematically selects and reviews both the academic and grey (blogs, web pages and white papers) literature on the security orchestration and automation domain, published between January 2007 and July 2017. This chapter provides a working definition of security orchestration and automation. It further classifies the main functionalities of SOAR platforms into three main areas: unification, orchestration and automation. We identify the core components of a SOAR platform and categorize the drivers of SOAR platforms based on technical and socio-technical aspects. This chapter also provides a taxonomy of SOAR platforms based on the execution environment, automation strategy, deployment type, mode of task and resource type. This chapter also highlights several areas of further research and development in the security orchestration and automation domains.

2.1 INTRODUCTION

Cybersecurity breaches lead to serious organizational and socio-economic consequences such as loss of revenue, damage to reputation and information systems, theft of proprietary data and customer sensitive information [47-50]. For example, in 2017 Equifax (one of the largest credit reporting agencies in America) [51] reported a major data breach that had affected around 148 million US consumers [52-54]. The hackers successfully stole sensitive information (e.g., credit card numbers, phone numbers, email addresses, and social security numbers) through that breach, which was preventable as per a recent report. According to research sponsored by IBM, the average total cost of a breach is around $3.92 million [50].

Organizations use various security tools to prevent known and unknown attacks and avoid the consequences that are associated with security vulnerabilities and threats [3, 16, 55]. Some of the commonly used security tools are antivirus, Firewall, Intrusion Detection Systems and Instruction Prevention Systems (IDS/IPS), and Security Information and Events Management (SIEM) [16-19, 55]. The security tools vendors use different technologies and paradigms to develop, deploy and operate their security tools, which cannot be easily integrated and interoperated for effective and efficient support of Security Operation Centers (SOC).

Security orchestration is aimed at introducing technical and socio-technical solutions to integrate multivendor security tools as a unified whole to support security teams in a SOC. Organizations are increasingly adopting SOAR platforms that are proactive, autonomous and collaborative solutions to enable security teams to perform their responsibilities effectively and efficiently [56-59]. A SOAR platform enables people, processes and technologies to work together to improve an organizations' security intelligence for better security operations and management [60-62]. Security orchestration is a prerequisite of security automation, which is the process of automatically detecting, preventing and recovering from cyberattacks without human interference, using information technology, automation algorithms and Artificial Intelligence (AI) tools and techniques [61, 63].

Existing security tools are designed to monitor an organization's IT infrastructures and network activities, generate security alerts and perform necessary actions upon

detection of security threats. An organization's security tools generate thousands of alerts, which are usually monitored and acted upon by security teams, mostly using manual or semi-automated processes and practices [63-65]. A Verizon's report indicates that 93% of data breach cases require minutes to be executed, but it can take companies weeks or months to discover attacks [66]. For example, after getting alerts from IDS for malicious behaviors, a security expert might go to an endpoint defense system to gather more relevant information by querying the network resources and validating a threat. After confirming the threat, a security expert commands a firewall to isolate or block the traffic from the affected region and update the threat information in the threat intelligence database. According to a report by Baker Hostetler [67], security experts took, on average, 61 days to discover the occurrence of an incident and, after discovery, 41 more days to take remedial action. A food chain, Wendy's, Point of Sale systems were affected by malware at 1025 locations in 2015, but it was first discovered in February 2016 [68, 69]. To deal with the potential threats of security breaches, security teams are expected to provision and facilitate the selection of the existing security solutions as quickly as possible to perform the required actions and ensure seamless security operation.

A SOAR platform has the potential to address the concerns of manual threat analysis, delays in responses to security incidents, as well as provide the security status of an organization's IT infrastructures. SOAR platforms are capable of automatically identifying suspicious activities in an organization's environment and proactively act to mitigate cyberattacks. According to a Gartner's report, by 2019, 30% of large and medium enterprises will be deploying some form of security orchestration and automation capabilities [70]. Another study [65] reports one third of organizations are planning to deploy or have deployed SOAR platforms that can bundle different security tools and human expertise together for the automation of security tools' activities within an organization.

Figure 2.1 captures some of the abovementioned organizational settings where several types of security tools generate alerts to be manually analyzed by security teams in the absence of a SOAR platform. A SOAR platform can automate most of these manual decision-making processes and accelerate incident responses by reducing the manual and repetitive activities. Orchestrating and automating the activities of

multivendor security tools requires a comprehensive view of SOAR platforms, as these tools have their own way to work and produce different formats of alerts. The existing SOAR platforms do not provide sufficient evidence of supporting different quality attributes such as flexibility, interoperability, scalability, modifiability, accuracy, integrability and extensibility [21, 58, 59, 71-74]. Given the increasing demand for security orchestration, a significant amount of research is needed to help understand the challenges in the existing solutions and practices of SOAR platforms to address the challenges.

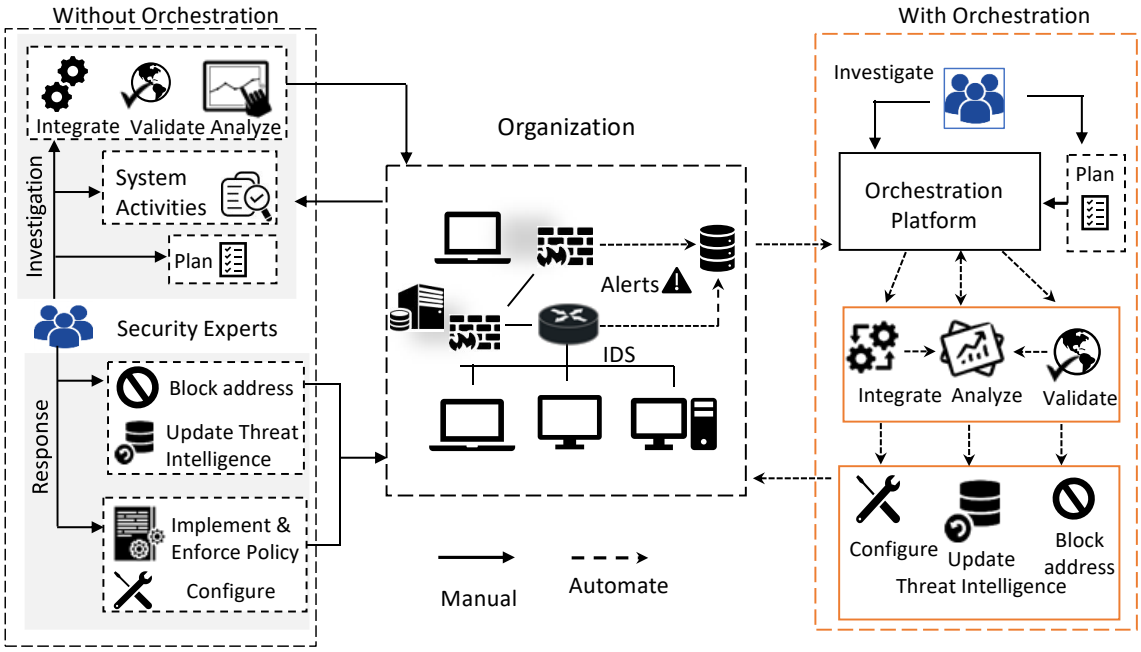


Figure 2.1 Overview of an organization's decision against alerts without security orchestration and with security orchestration

This chapter reports a Multi-Vocal Literature Review (MLR) which aims to systematically identify and review the literature on security orchestration under the conditions, "state of the art" and "state of the practice". An MLR (i.e., a type of Systematic Literature Review) includes both peer reviewed and non-peer reviewed literature (e.g., newsletters, white papers, fact sheets, and blog posts) [46, 75]. A Systematic Literature Review (SLR) has become the most popular method of conducting a literature review in Software Engineering (SE) [76].

An SLR focuses only on peer-reviewed literature and does not include grey literature. An SLR may not always provide an established discipline of knowledge, as it

ignores a large amount of information produced by software engineering practitioners [46, 75, 77]. Hence, the MLR is attracting more attention in SE [46, 78]. We believe that conducting an MLR in the area of security orchestration will be more useful than an SLR as there is a large body of non-peer reviewed literature reported by practitioners. We conducted this MLR to explore the fundamental challenges and opportunities for the evolution of SOAR platforms. We analyzed the characteristics of existing SOAR platforms to understand how to solve the challenges associated with security orchestration. We also investigated the strengths and weaknesses of the technologies used in SOAR platforms. The main contributions of this chapter are:

- It introduces a working definition of security orchestration, followed by several functionalities of SOAR platforms ranging from integrating several security tools to performing incident response planning against a threat, as well as enabling collaboration among security tools to materialize the concept of security orchestration (refer to sections 2.3 & 2.4 for further details)
- It identifies the key challenges that practitioners and researchers intend to overcome through security orchestration (details are discussed in section 2.5).
- It provides a taxonomy of different aspects of SOAR practices that are needed to support the dynamic adoption of applications in an organizations' environment (details in section 0).
- It determines and discusses the open research challenges and issues in the field of security orchestration (refer to sections 2.7 & 0).

2.2 RESEARCH METHOD

The methodology used for this chapter has benefited from the SLR guidelines reported in [79]. The methodology adopted to carry out the MLR was inspired by the work reported in [80, 81]. It involved three main phases: (i) planning and designing the review protocol, (ii) conducting the review and (iii) reporting the review. We developed a review protocol describing each step of an MLR. The review protocol included several steps: research identification, search strategies, study selection, data extraction and synthesis. Our MLR process follows the steps in the same order as shown in Figure 2.2.

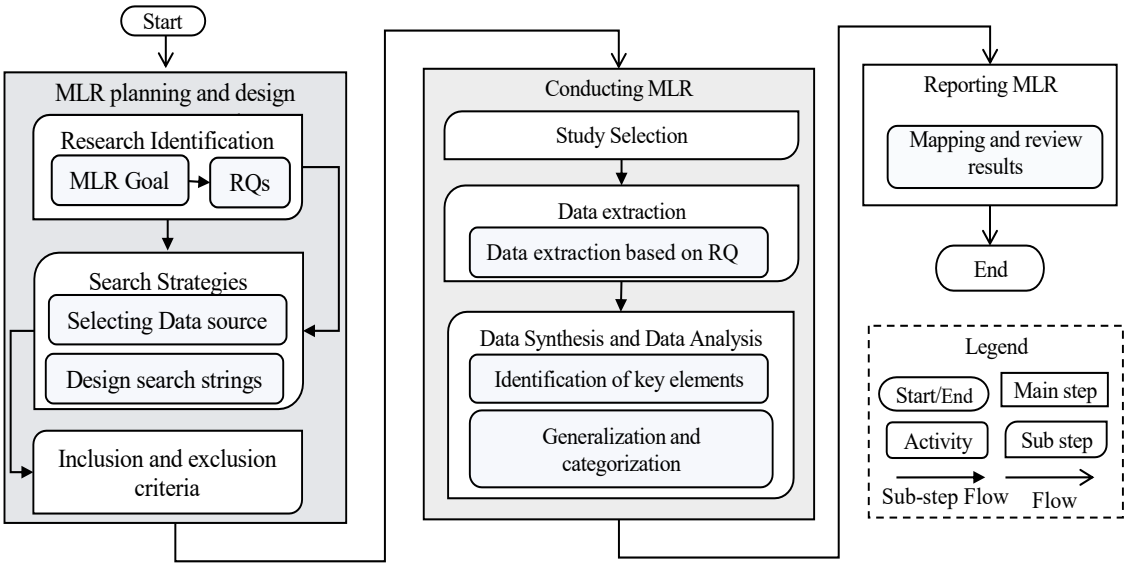


Figure 2.2 An overview of our MLR process

2.2.1 Research Identification

We identified the relevant literature using a search strategy that was based on a set of research questions, as shown in Table 2.1. The research questions purported to help gain an understanding of SOAR platforms, their functionalities and respective elements. The research questions were also aimed at helping to identify and review the supportive tools, approaches and evaluation criteria for adopting a SOAR platform in practice.

2.2.2 Search Strategy

The following sections detail the search strategy for acquiring the relevant literature from multiple sources.

2.2.2.1 *Data Sources*

The review includes both peer reviewed and grey literature that was identified and acquired using both manual and automatic searches in the relevant sources. Initially, we performed a manual search on the *Journal of Computer Security*, *ACM SIGSAC Conference on Computer and Communications Security*, and *USENIX Security Symposium* to gain an overview of the recent literature. We also searched the recent proceedings of RSA conferences. Then we conducted an automatic search in three digital libraries, *IEEE Xplore*, *ACM Digital Library* and *Scopus*, that publish peer reviewed literature on computing.

Table 2.1 Research Questions of this MLR

Research Questions	Motivation
RQ1. What is Security Orchestration?	The first question (RQ1) investigates how security orchestration is defined. RQ1 aims to identify the relevant related work, i.e., identifying keywords for the literature search that lead to maximum coverage of the related approaches. RQ1 also investigates the functional features and core elements of security orchestration.
RQ2. What challenges do security orchestration intend to solve?	Security orchestration is commonly used by practitioners to bring automation, streamline incident response and integrate security tools. Many challenges that exist for security tools in a more traditional setting also apply to security orchestration. RQ2 focuses on the aspects where security orchestration fundamentally differs from traditional approaches.
RQ3. What types of solutions have been proposed? RQ3.1. What practices have been reported for adopting security orchestration? RQ3.2. What types of tools and techniques do researchers and practitioners use, propose, design and implement in practice? RQ3.3. What aspects of architecture security do practitioners consider for large-scale deployment of security orchestration?	The motive behind this question is to identify the solutions related to security orchestration and the reported practices followed by organizations (i.e., requirements, guidelines and collaborative approaches) for adopting security orchestration (RQ3.1); more specifically how existing tools and techniques are employed to implement a SOAR platform; what are the innovative approaches and techniques needed for successful implementation of a SOAR platform (RQ3.2); and, most importantly, what aspects of architecture are being considered for large-scale deployment of a SOAR platform (RQ3.3). RQ3 would help researchers to find the gap and practitioners to consider the architectural aspects that need to be considered to successfully implement a SOAR platform on a larger scale.

We used the advanced search option to facilitate a type of search that enables a multiple keywords search. During the automatic search in the digital libraries, we defined the search to match the search string with the titles, abstracts and keywords of the papers published between January 2007 and July 2017. Our search in *ACM DL, IEEE Xplore*, and *Scopus* included papers from the *Annual Computer Security Applications Conference*

and *IEEE Security and Privacy*. In addition, we searched in Google Scholar to search and include some relevant literature, especially some patents that we could have missed through the abovementioned search procedure.

To search the grey literature, we used the Google search engine, like other MLRs [77, 78]. We search the first ten pages, which are considered sufficient to find the most relevant literature, as Google's search engine's algorithm retrieves and shows the most relevant results in the first few pages [77, 78]. For example, Google's search engine returned 45,900 results for the term "security orchestration" in November 2017; however, the relevant content was captured in the first ten pages.

2.2.2.2 *Search Strings*

We created a search string to ensure a thorough search over several databases. For academic literature, we formulated the search string based on a) the key terms gathered from the relevant papers, b) synonyms, alternative terms and related concepts of security orchestration, c) **AND** and **OR** to combine all the terms. We performed several pilot searches and refined, discarded and added search terms to confirm the inclusion of the relevant papers that we already knew. We formulated the search string in three parts. We performed a match of the search string with the paper's titles, abstracts and keywords. We used the following search string.

Search String 1	("Security" **OR** "Alert" **OR** "Threat" **OR** "Policy" **OR** "Intrusion" **OR** "Anomaly Detection" **OR** "forensic") **AND** ("Orchestration" **OR** "Instrumentation" **OR** "Coordination" **OR** "Correlation" **OR** "Collaboration" **OR** "Automation" **OR** "Integration") **AND** ("Security Tool" **OR** "Safeguard Software" **OR** "IDS" **OR** "IPS" **OR** "Threat Intelligence" **OR** "Detection Engine" **OR** "Prevention Engine" **OR** "Security Control" **OR** "Security Appliance" **OR** "perimeter defense" **OR** "Incident Response")

We used the search string *"Security **AND** Orchestration"* to search the grey literature and conducted a search using the Google search engine and Google Scholar.

2.2.3 Eligibility Criteria

We defined a set of inclusion and exclusion criteria to select relevant papers. The criteria are shown in Table 2.2. Since this review is a blend of scientific and grey literature, we used narrow inclusion and exclusion criteria.

Table 2.2 Inclusion and exclusion criteria

Inclusion criteria	Exclusion criteria
• IC1: Articles in English and full text is accessible. • IC2: Articles that focus on developing integrated, coordinated and collaborative solutions. • IC3: Articles include a sound validation (for grey source: working prototype or tools, proper references to validate the result). • IC4: Articles that reports practices and challenges in cyberspace (such as blogs, magazine and reports) to give an indication of security orchestration.	• EC1: Short academic paper (paper less than 6 pages). • EC2: Articles whose focus is irrelevant to security. • EC3: Articles that focus on physical infrastructure or hardware. • EC4: Articles where the focus is to enhance algorithms or features of a single security solution. • EC5: All duplicate articles found from various sources.

2.2.4 Study Selection

Figure 2.3 shows details of the selection of grey and academic literature at each step of the MLR. This also includes the search databases and number of papers selected after each step. We followed two different approaches to select the academic and grey literature.

2.2.4.1 *Selection of Academic Literature*

In this section, we describe each step of the process that we have followed to select the relevant papers. Our search in *ACM DL*, *IEEE Xplore* and *Scopus* returned 271, 600 and 1017 results, respectively. The titles, abstracts and keywords of these papers were examined. For some papers, just reading the title and abstract was not enough to decide whether to keep them in the selected papers' pool. We kept those papers for the next round. A total of 1617 papers were discarded based on the inclusion and exclusion criteria described in Table 2.2. We read the title, abstract and keywords of each paper in the *Journal of Computer Security*, *ACM SIGSAC series of Conferences on Computer and Communications Security* and *RSA series conferences* and filtered 19 papers.

After round 1, we selected 290 papers. Then we removed the duplicates and excluded the papers that were shorter than 6 pages. Finally, we screened the whole text and applied the eligibility criteria to select the relevant papers. A total of 37 papers were

selected from the digital libraries. To ensure the inclusion of any relevant papers that we might have missed, we extended the search in Google Scholar. We searched for the string "Security and Orchestration" and checked both the titles and abstracts of the top 200 results. We only included 6 articles that were not found in the automatic and manual search procedures of phase 1. We applied all the eligibility criteria while selecting the papers from Google Scholar.

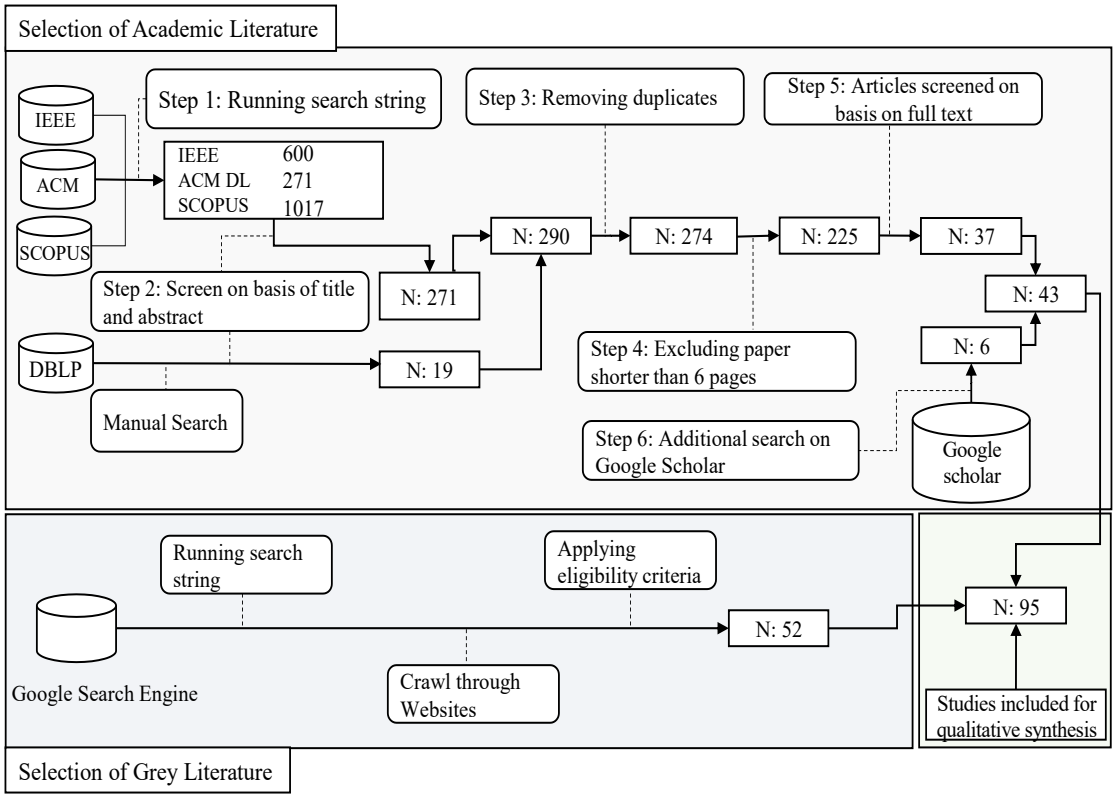


Figure 2.3 Study selection process of our MLR

2.2.4.2 *Selection of Grey Literature*

In the next phase, we used the Google search engine and checked the first 10 pages. We only continued further if needed. We identified several practitioners (niche and start-up) who were contributing to the field of security orchestration. We crawled through their websites and looked for the relevant resources and white papers. We applied the eligibility criteria while selecting the papers. At the end of this process, we identified a total of 52 papers, including white papers, blogs, news articles and websites. Finally, we included 95 pieces of literature (Figure 2.3) for data extraction and synthesis. Figure 2.4 showed the distribution of the selected pieces over several types of venues. For both

cases, we excluded those papers published before 2007. Table 2.3 lists the papers that were finally reviewed.

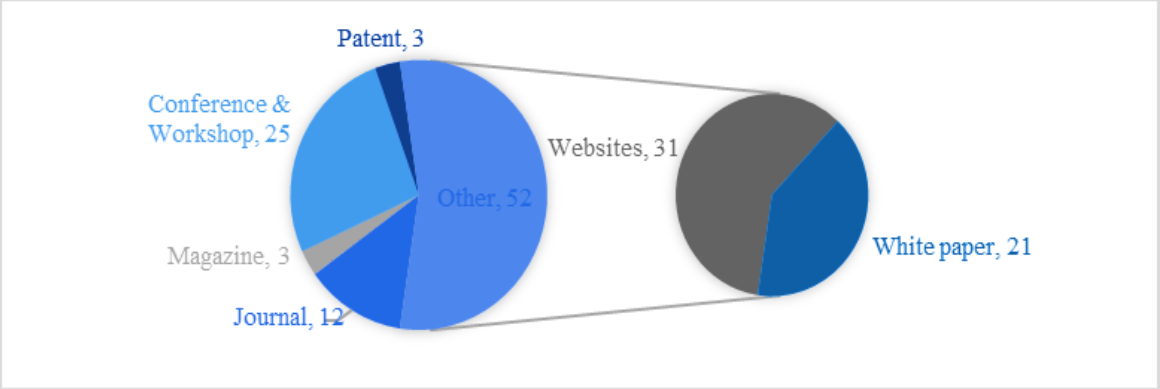


Figure 2.4 Distribution of selected articles over venues

Table 2.3. Study selected for data extraction and qualitative analysis

Academic Literature	**Grey Literature**	
	Websites & Blogs	**Whitepapers**
[21, 23, 58, 59, 72-74, 82-117] = 43	[17, 19, 20, 32, 33, 35, 56, 57, 60, 62-65, 118-135] = 31	[18, 24, 29, 61, 71, 136-151] = 21

2.2.5 Data Extractions, Synthesis, and Analysis

Following the process of MLR, at this step, we read, assessed, extracted data and summarized the findings from the selected papers, based on the pre-defined RQs and motivations (Table 2.1).

2.2.5.1 *Data Extraction*

We identified and extracted the relevant data using a pre-defined data extraction form from each of the selected sources. We needed these data to answer the research questions. We also extracted some general information, e.g., author's name, venue published and published year. We conducted a pilot study on a set of 10 sources before deciding how to extract the required data. We stored all the extracted data in a spreadsheet for analysis.

2.2.5.2 *Synthesis and Analysis*

The extracted data were stored in three different sections of the data extraction form to perform the synthesis and analysis of the extracted data. These sections are (a) security orchestration definitions, functionalities and applications, (b) challenges to be solved and (c) security orchestration practices, tools and techniques. We analyzed each set of the data items using qualitative analysis methods. We used a combination of different qualitative analysis methods (i.e., narrative synthesis and thematic analysis). For example, for classification and categorization of data, we used thematic analysis [152, 153]. We followed several steps to analyze the data, including getting familiar with the extracted data by carefully reading each piece of data. We collaboratively analyzed and systematically synthesized the extracted data to develop a taxonomy to report the results in a generalized form. The taxonomy developed in this way has been used for reporting the functionalities, benefits and aspects of security orchestration in this chapter. For data analysis, we followed the qualitative data analysis guidelines [153]. Table 2.4 contains the abbreviations that are used in this chapter. We report the synthesized result in sections 2.3, 2.4, 2.5, and 0.

Table 2.4 Summary of Notations

Acronym	Abbreviation/ Description	Acronym	Abbreviation/ Description
AI	Artificial Intelligence	MLR	Multi-vocal Literature Review
AIRS	Automated Incident Response Solution	NFV	Network Function Virtualization
API	Application Programming Interface	NTT	Nippon Telegraph and Telephone
DBot	Demisto's chatbot	NVD	National Vulnerability Database
DDoS	Distributed Denial of Service	OADS	Orchestration-oriented Anomaly Detection System
DHCP	Dynamic Host Configuration Protocol	OVAL	Open Vulnerability and Assessment Language
DNS	Domain Name System	ROI	Return on Investment
DXL	Data Exchange Layer	RQ	Research Question
EC	Exclusion Criteria	SDDC	Software Defined Data Centre
ETSI	European Telecommunication Standard Institute	SDN	Software Defined Network

Acronym	Abbreviation/ Description	Acronym	Abbreviation/ Description
IC	Inclusion Criteria	SDSec	Software Defined Security
IDS	Intrusion Detection System	SE	Software Engineering
IDMEF	Intrusion Detection Message Exchange Format	SIEM	Security Information and Event Management
IOC	Indicator of Compromise	SLR	Systematic Literature Review
IPS	Intrusion Prevention System	SOSDSec	Service-Oriented Software-Defined Security
IRP	Incident Response Plan	STIX	Structured Threat Information Expression
IT	Information Technology	TAXII	Trusted Automated Exchange of Intelligence Information
LEEF	Log Event Extended Format	PSI	Premise-aware Security Instrumentation
MD5	Message Digest 5 algorithm	VNF	Virtual Network Function

2.3 SECURITY ORCHESTRATION: DEFINITIONS, FUNCTIONALITIES, AND ELEMENTS

This section presents the findings for RQ1 *"What is Security Orchestration and Automation?"*. Our data analysis for RQ1 reveals some key definitions of "security orchestration" given by practitioners, the functional and non-functional requirements and the key functional components of a SOAR platform.

2.3.1 Definitions

Our analysis shows that practitioners use the term security orchestration widely, with no clear and common definition. We assert that having a common working definition of security orchestration and automation will help practitioners and researchers to define a discipline of research and practice for promoting practices, processes, tools and technologies. The term security orchestration is being mostly used as a buzz-word that can lead to misinterpretation of the core concept of orchestration [60, 120, 121, 137]. Some organizations and practitioners confuse security orchestration with security automation [65]. We present a few key definitions from the reviewed work.

According to HEXADITE, "*Orchestration is the practice of connecting existing security tools together through APIs to streamline incident response processes.*" Here, Hexadite has considered security orchestration more as an integration tool and presented

a definition for security automation. Barak Klinghofer, CPO of HEXADITE [19] has defined "*The active process of mimicking ideal steps a human would take to investigate a cyber threat, determining whether the threat requires actions, performing necessary remediation actions, deciding what additional investigation should be next*" as security automation [139]. According to a start-up company, KOMAND [17], security orchestration is more than just connecting security tools. Their definition [17] is:

> "*Security orchestration is a method of connecting security tools and integrating disparate security tools. It is the connected layer that streamlines security processes and powers security automation.*"

Markets and Markets [126] stated that "*Security orchestration is an approach to automatically respond to security incidents and protect IT systems in organizations from advanced cyberattacks and vulnerabilities*".

Microsoft has distinguished security orchestration and security automation [63] as:

> "*Security orchestration is the integration of security and information technology tools designed to streamline processes and drive security automation*" and "*Security Automation is the use of information technology in place of manual processes for cyber incident responses and security event management*".

ThreatConnect [122] has also presented distinct definitions for security orchestration and security automation:

> "*Security orchestration is the connecting and integration of various security applications and processes together" and "Security automation is the automatic handling of a task in a machine-based security application that would otherwise be done manually by a cybersecurity professional*".

ThreatConnect has defined security automation and orchestration [122] as: "*Security automation and orchestration is a coordination of automated security tasks across connected security applications and processes.*"

According to Forrester, security orchestration and automation should be described together as technology products. They have defined security orchestration and automation [120] as:

> "*Technology products that provide automated, coordinated, and policy-based actions for security processes across multiple technologies, making security operations faster, less error-prone and more efficient*."

Clearly, Forrester's definition asserts that automation can help take the full benefits from security orchestration. Bruce Schneier [56], chief security officer of IBM, defines security orchestration as the unification of people, processes and technology. He claims that security orchestration is keeping people in the loop of security automation, where the computer performs the automation of certain activities, but a human coordinates the activities. It is more about making people effective. He also pointed out that the security incident response needs to be dynamic and agile. DFLabs' Oliver Rochford has defined security orchestration as the junction where people, process and technology all come together [60]. According to him, people build automation into the process and consume information and insight generated by technology. Security orchestration and automation is the realization of three paradigms – unification, orchestration and automation. Our definition of security orchestration is as following:

> "*Security Orchestration is the planning, integration, cooperation and coordination of the activities of security tools and experts to produce and automate required actions in response to any security incident across multiple technology paradigms.*"

This definition provides siloed security tools the ability to share information and threat intelligence among them through an integrated and unified platform. This is achieved by seamless monitoring, situational awareness, data analytics, knowledge representations and semantic knowledge sharing across the existing security tools. A SOAR platform works as an intelligence assistant for security experts. It is clear that there is a need for extensive training to learn from human behavior to provide AI capabilities that can enable an enterprise to achieve long-lasting development and deployment of a SOAR platform using existent tools and protocols.

2.3.2 Functionalities of Security Orchestration and the Automation Platform

In this section, we report the findings of the functionalities of SOAR platforms. Several reports (i.e., References [120, 136]) have mentioned security orchestration as one of the

emerging technologies, which has the potential to be widely adopted in near the future [136]. One of the motivators of security orchestration is to bridge the gap between detection and remediation of security incidents [19, 144]. Most of the detection solutions are automated, whereas the response processes are still reliant on humans. To bridge this gap, there is a need to unify the activities of security tools, streamline the workflows and choose the right courses of action. According to Demisto [148], a comprehensive SOAR platform must be able to automate security tools' activities, create playbooks with complicated logic, and track and orchestrate the tasks assigned to an analyst. Paul Weeden [129] has stated: "*Security orchestration makes the most of human skills by bringing together automated tools and reports to provide risk information exactly when and where it is needed.*"

Figure 2.5 highlights the key functionalities of a SOAR platform in three paradigms, (i) unification where the security orchestration acts as middleware, (ii) orchestration that is the process of translating complex processes into streamlining workflow and (iii) automation that enables an automated response.

2.3.2.1 ***Middleware/ Hub***

Vendors have mentioned security orchestration as a platform that acts as a hub for unification, coordination, data sharing and analysis for disparate cybersecurity and IT technologies [18, 58, 141]. Security teams can easily integrate multivendor security tools, share threat intelligence and collaborate with the external organizations to gain an insight into an organization's security state through a SOAR platform. SDSec (Software Defined Security services') orchestration solution has a layer of functionalities to perform communication and coordination across different subsystems [21].

Unify security tools*:* Several of the reviewed works mentioned that a SOAR platform unifies disparate security tools and processes [17-19, 29, 65, 72, 101, 154], integrates the enterprise's security architecture [[136, 148], connects detection, networks and endpoint security tools [59, 144], and performs coordination among security tools' activities [58, 61, 65, 83]. Connecting the activities of disparate security tools makes the incident handling process efficient and effective for the security team. A SOAR platform provides a single console or platform to integrate security tools' activities, removes the operational silos and helps security teams to free their time [128, 143].

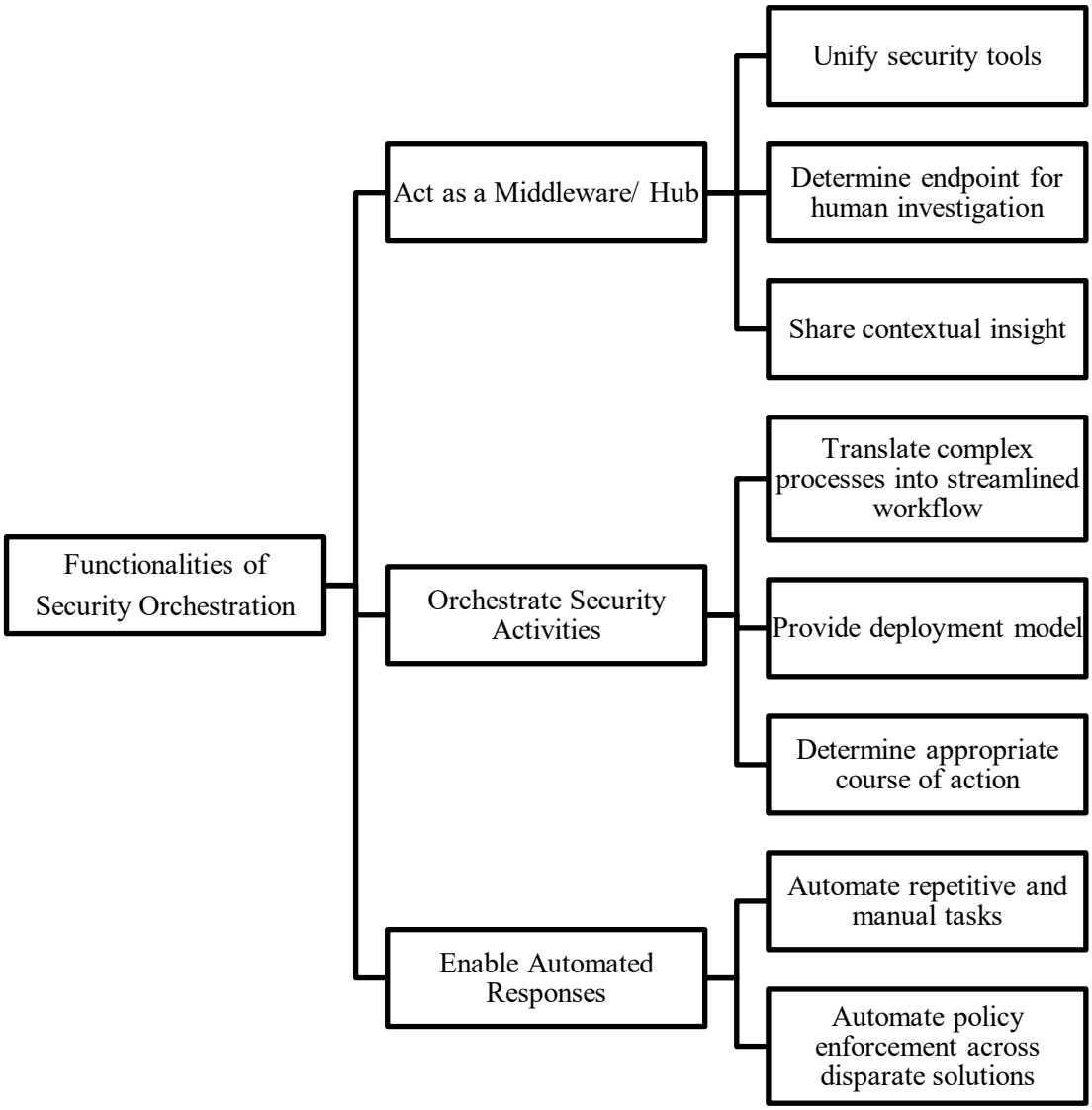


Figure 2.5 Key functionalities provided by a SOAR platform

Unifying intelligence according to vulnerability also minimizes the overall complexity of the incident response process [72, 83, 92]. Through the coordination platform, security tools can interoperate with each other to enhance organizational protection and defense systems [101, 130, 145]. Feitosa et al. [23] have proposed collaborative solutions to detect intrusions and anomalies by analyzing the co-creation of events and alerts among different subnetworks. The orchestration tools proposed in Reference [58] are designed to coordinate the safeguarding function by calling an individual software package with respect to an installation. Jeong et al. have proposed a coordination module for organizational architecture integrating cyber forensic functions [117].

Determine endpoint for human investigation: A SOAR platform [120] can enable security teams to gain insights into several security tools' activities, operate disparate security tools as a unified system [59, 61] and collaborate with other security teams for planning and decision making [17, 128, 130]. A SOAR platform informs and educates security teams about an organizations' threat behaviors [120] and notifies about the supported policies [144]. By orchestrating various activities, the platform can decide when human insight is required [17, 19, 24, 114, 121, 137]. For example, the work in [82] has highlighted the critical assets with high priority to the administrator for investigation. The motive is to keep the security team focused on threats that demand their immediate attention and expertise.

Share contextual insight: A single security prevention and detection system usually suffers from the tunnel vision syndrome that leads to an inability to detect certain types of security attacks, such as Distributed Denial of Service (DDoS). A SOAR platform gathers threat intelligence from various external sources (e.g., web pages and blogs), extracts key features from a huge volume of threat intelligence data and provides the contextual insight related to alerts or attacks to a security team. In addition, it engages security tools to perform complete monitoring of endpoints [93, 136, 145], correlates their activities and provides real-time visibility of known and unknown threats to security teams [29, 128, 131]. An organization can share contextual device data with the third-party system [18, 144]. It helps security teams reduce and mitigate risk exposure [29, 144], make a faster decision based on context [24, 35, 59, 114] and gather an overview of what is happening in various subnetworks within an organization [18, 61, 62, 136]. By sharing the contextual insight, a SOAR platform works as a collaborative platform that also enables training of security teams based on past investigations [35, 61, 128]. The online evaluation framework proposed in [82] has provided situational awareness to an organization so that it can take appropriate actions. By assessing the security state of an organizations' different assets, the proposed framework helps the administrator to identify compromised assets and prioritizes alerts [82]. A set of papers [72, 86, 88, 94] has proposed a platform for security teams and security tools used in an organization to share their knowledge. Jeong et al. [117] have followed the structure of having a coordination group with a participant group to propagate the relevant information to the external work or another coordination group. RiskVision has proposed SOAR platforms

to unify stakeholders in business, IT and security tools to provide automation for end-to-end cyber risk prevention and response [130].

2.3.2.2 *Orchestrate Security Activities*

Translate complex processes into streamlined workflow: After receiving alerts, security experts need to perform multiple steps to find the attacks, vulnerabilities, affected endpoints and mitigation solutions. These steps include the complex process of data collection, investigation, remediation, evaluation of actions and deciding on the appropriate courses of action. Several papers [17, 19, 65, 71, 82, 122, 139, 141, 148] have mentioned that the motivation for orchestration is to translate the complex process of threat investigation into a streamlined workflow through automation and orchestration. A streamlined workflow requires a standardized process that includes proper planning for incident responses, policy executions, investigations, response actions and remediation processes [21, 29, 140, 143]. The workflow is designed to mimic security teams' activities during threat investigation to reduce the cumbersome manual processes, human errors and improve security teams' capabilities for incident responses.

Orchestrating and integrating security tools' activities allows security teams to simplify complex workflows, coordinate the flow of data and tasks and enables powerful machine to machine automation [17, 29, 137, 151, 154]. The task can be fully or partially automated, based on the complexity of the threats [143]. ForeScout has proposed a rule engine and a workflow engine to make instant decisions and offered data aggregation to provide in-depth awareness about the organizational environment [136]. The online Evaluation framework, Seclius [82], has translated alerts into system security measures to reduce the reliance on human expertise in capturing system characteristics through low-level alerts. This work also provides a ranking of the affected system's assets and malicious events for organizations to help the security administrator [82]. Similar to this, the Premise-aware Security Instrumentation (PSI) policy engine proposed in Reference [74], has translated the high-level security postures provided by an administrator into per device intents.

Provide a deployment model: Several security vendors provide SOAR platform deployment services that require appropriate orchestration and automation of existing security tools, along with organization external and internal infrastructures [32, 87, 90,

118, 124]. FireEye deployment service for automation provides functionality to manage events across multiple FireEye and third-party products, and ensures deployment is successful [118]. The deployment model depends on the organizations' scale, complexity and course of actions. Vendors are providing flexible deployment models for organizations to ensure simple installation and management of various forms of infrastructure. This action also ensures efficient deployment in heterogeneous environments. Organizations can choose security policies based on their need to restrict access and tailor security configurations [71, 146]. Additionally, testing and evaluation of deployment models are also undertaken once any change has been made. The proposed system can provide a progressive deployment module to perform upstream rules filtering that helps to reach the source of attacks [86]. The work in Reference [87] has proposed an innovative solution to perform quick deployment of various security mechanisms. The orchestration system proposed by Reference [90] has arranged appropriate virtual instances in the right place: virtual appliances are automatically added and controlled. It has also automatically moved traffic to virtual networks to prevent major harm, block the attack and strengthen the system.

Determine appropriate courses of action: A SOAR platform can help to resolve an incident promptly to determine the appropriate and effective course of actions [19, 21, 32, 118, 127, 128, 148]. By choosing the appropriate course of actions, a SOAR platform maintains process consistency across a security program. FireEye has mentioned the in-built course of actions with automated support for all the needed steps for handling a security incident as a core of a SOAR platform [118]. Also, various kind of alerts (i.e., phishing and endpoint contamination) are needed to distinguish remediation activities with different courses of actions. Upon investigating an alert, a SOAR platform can determine the proactive response to threats or may initiate an additional investigation based on an attack's complexity [118, 121, 144]. In many cases, a post-attack investigation or evaluation task can also be instantiated. Feitosa et al. [23] have proposed a framework, *"Orchestration-oriented Anomaly Detection System (OADS)"*, that performs coordination and collaboration among different anomaly detection techniques to detect and evaluate threats and choose right actions. Security teams do multiple investigations in response to an alert. In the process of orchestration, one investigation usually triggers multiple investigations [19].

Workflows are designed to choose the appropriate course of actions, simplify threat responses through integration and automation [17, 18, 29], perform necessary remediation [144], decide additional investigation(s) [139], design documents for playbook review [118] and define sources of information to help the security team to solve the identified problems [142]. A formal workflow helps maintain effective communication and strong collaboration among security teams [139]. Providing a formal workflow helps security teams and SOAR platforms to maintain consistency across actions [29]. This simplifies and accelerates alerts investigations, eases proactive hunting of attackers, accelerates the Return on Investment (ROI) and eliminates the need for continuous assessment. The security functions are inserted dynamically into the workgroup, based on the policies [71]. The operating principle of the framework in Reference [59] has security tools that operate on their own and perform activities of context sensing, policy decisions and policy enforcement. Each activity is logically independent of each security tool. The SOAR tool proposed in [58] works as an interface to perform security scanning and testing or other security activities. This tool enables different safeguard software packages to come to an agreement for invoking the necessary activities(s) owned by any of the software packages. The framework discussed in Reference [117] helps to implement the coordination of an organization's activities effectively by performing an on-site and online investigation, to provide security warnings and appropriate response actions.

2.3.2.3 *Enable Automated Responses*

A SOAR platform automates incident response activities. HEXADITE has automated 800,000 man-hours of work in two years that is equivalent to US $38.5 million in customer savings [19]. Several papers [18, 19, 35, 136] have reported that a SOAR platform automates the entire threat defense life cycle and provides intelligence automation services. ETSI (the European Telecommunication Standard Institute) considers automating the control of deployment and configuration of the security tools' activities as a substantial prerequisite of orchestration [91]. According to Forrester, orchestrating incident response activities enables an automated response without the need for coding skills [120]. A SOAR platform allows autonomous control and protection of the network through discovered insights [144].

Automate repetitive manual tasks: SOCs use a SOAR platform to automate repeatable tasks and remove duplicate incidents to optimize security teams' capabilities and reduce the overall cost [17, 29, 35, 63, 125, 131, 144, 146, 154]. Automating the routine tasks helps security teams to tackle more critical problems [17, 18, 65, 122, 139, 144]. According to Swimlane, 80% to 90 % of all security operations of an incident response can be automated to some extent [150]. The collaborative incident response planning process design discussed in Reference [85] helps practitioners to come up with a repeatable and executable planning process. Some vendors provide a SOAR platform that also automates the deployment of security tools' functions through a network infrastructure [71]. Several papers [72, 87, 121, 130] have proposed automation of the analysis of cyber threat intelligence, which includes extracting data from technical blogs, websites, finding correlations among different reported attacks and updating incidents' severity levels based on threat intelligence feeds. A SOAR platform reduces the response time by minimizing the error-prone manual process and codifying real-world expertise. Koyama et al. have also reported a security operations automation framework that helps in optimizing decisions with regards to a variety of security sensors and appliances (i.e., security tools) [72]. Luo et al. [21] have proposed automated cybersecurity operations in the software-defined environment.

Automate policy enforcement: Security policy enforcement greatly benefits from automation that considers all the tools, devices, and measures required for security policy implementation. A SOAR platform enables an organization to automate policy enforcement and configuration at runtime [71, 73, 74, 91, 99, 101, 141]. A set of systems developed to derive policy decisions based on contextual data and provide real-time policy enforcement have been reported in References [59, 73, 99]. Yu et al. [74] have introduced a multistage mapping mechanism to adjust policies automatically, based on network devices. In Reference [73], policy enforcement is performed automatically. ETSI has aligned security policies in an automated way inside virtual, physical and hybrid networks [91]. Dynamic enforcement of policies allows automatic configuration of security tools and updates of threat intelligence [71]. An organization can automate policy enforcement across disparate solutions [144]. Luo et al. [21] have proposed the provision of consistent security policies by orchestrating software-defined security services across a heterogeneous cloud environment. A generic SOAR framework

proposed in Reference [59] enables ad-hoc and context-aware policy criteria to be applied in real time by using an ecosystem of security tools connected via a Data Exchange Layer (DXL).

2.3.3 Quality Requirements for Security Orchestration Platforms

We have identified the key quality requirements/ Non-Functional Requirements (NFR) (also known as quality attributes) of a SOAR platform. Figure 2.6 shows the main quality attributes of a SOAR platform gathered from the existing literature. These are the non-functional requirements of a SOAR platform. Every organization needs to consider these attributes before adopting or implementing a SOAR platform. The quality attributes of a large-scale system are expected to guide the key architecture design decisions. Hexadite has mentioned the pre-requisite for a SOAR platform, which is basically the quality attributes of such a platform [140].

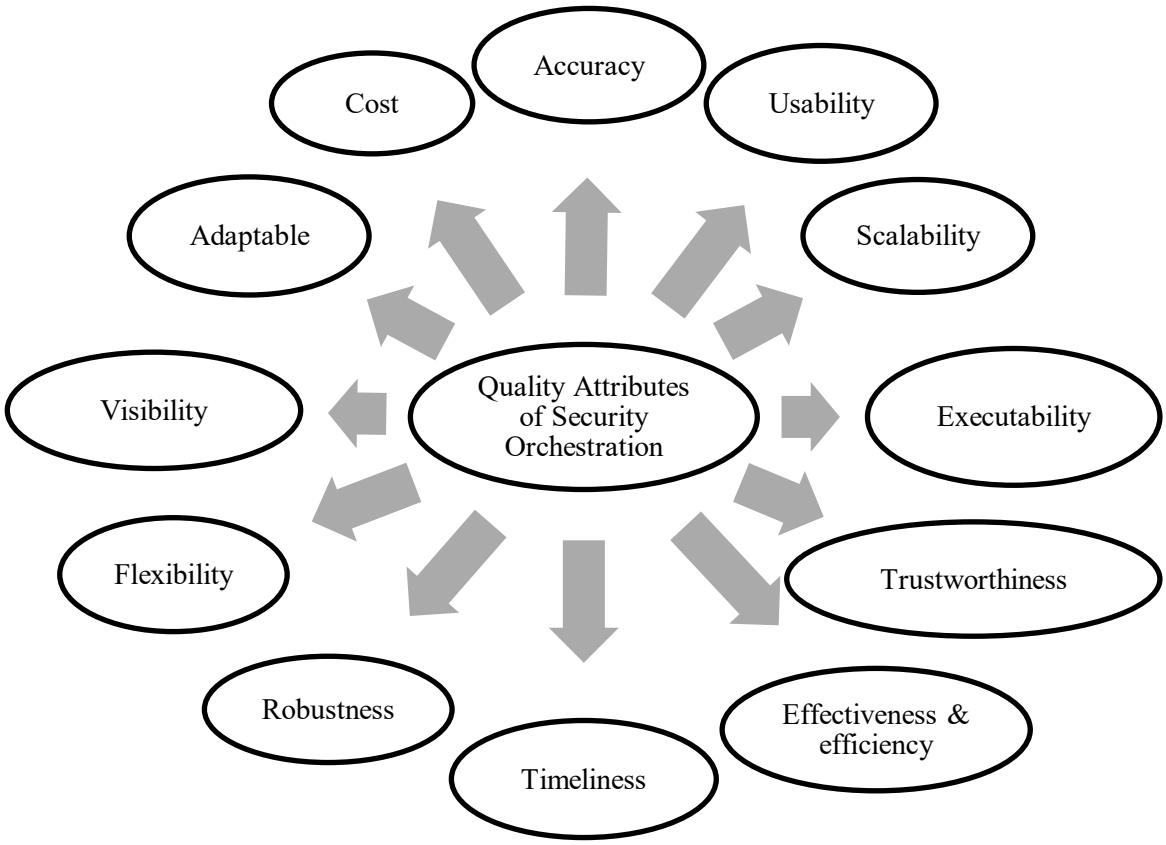


Figure 2.6 Quality Attributes of a Security Orchestration Platform

Table 2.5 lists the quality attributes and a set of corresponding metrics that can be used to measure these attributes. For example, usability is a crucial factor for the effective utilization of a SOAR platform. A SOAR platform requires a simple and powerful user interface that can be easily customizable for different types of SOAR platform users. A SOAR platform should have a flexible architecture, so that each security team can create a work environment according to their need for a service.

Table 2.5 Quality attributes of a SOAR platform

Quality Attribute	Measurement Metrics	Articles
Accuracy	- Accuracy of diagnosis - Appropriate measure against attacks - Accurate classification and reliable taxonomies of threats - Data integrity	[23, 59, 72, 82, 83, 87, 89, 95, 98, 101, 102, 105, 106, 113, 130, 136]
Usability	- Ease-of-use, easy to manage, connect and repeat tasks and interruptible - Simplified user interface to control security tools - Simplification of security management tasks for the network administrator and auditing module - User satisfaction - Higher analyst productivity - Accessible and stable threat intelligence	[19, 21, 24, 35, 58, 61, 82, 84, 85, 90, 92, 104, 112, 114, 122, 137, 140]
Scalability	- Vendor agnostics - Independent security policy orchestration - Extensible architecture	[19, 21, 24, 71, 73, 82, 84, 86, 88, 102, 104, 112, 121, 124, 136]
Executability	- Qualitative and quantitative information about security incidents - Measurable security tools - Measurable goals - Security state of different assets of organizational infrastructure	[17-19, 24, 64, 85, 92, 113]
Trustworthiness (Reliability)	- On human: expertise level, fairness to collaborator and reputation - On existing security tools: trust value and predictability	[23, 59, 72, 82, 83, 85, 87-89, 92, 95, 98, 101, 102, 105, 117]

Quality Attribute	Measurement Metrics	Articles
Effectiveness and Efficiency	- Increase in detection rate - Fewer overheads for the security team - Reliance on human ability and satisfaction - Optimizes resources and performance - Predictable cost structure and indicator of compromise; - Key indicator to measure security effectiveness: Mean time to notification, remediation and investigation - Speed of integration and speed of deployment	[18, 24, 35, 59, 61, 71, 82, 85, 93, 97, 98, 101, 104, 106, 113, 116, 127, 136, 140, 150]
Timeliness/ Speed	- Time to perform raid recovery - Time to detect, triage attack and remediation - Time need to analyze an attack - Time for policy enforcement - Delay in business activities - Overall latency of packet processing	[18, 19, 23, 24, 59, 61, 62, 70, 72, 84-86, 90, 101, 104-106, 112, 114, 121, 122, 125, 136, 140]
Robustness	- Robustness to DDoS - Capacity for attack detection - Incident response capacity	[18, 24, 73, 86, 88, 100, 101]
Flexibility	- Feasible to update - Flexibility to design workflow automation - Flexibility to adapt process and accelerate response to all types of threats	[21, 23, 24, 35, 59, 62, 73, 90, 102, 104, 121, 125, 131, 136]
Visibility	- What analyses are available and clear definitions of their capabilities - Security state of an organization - Secure configuration guidelines	[18, 19, 21, 24, 35, 59, 71, 89, 92, 94, 103, 125, 130]
Adaptable	- Compatibility with existing network topology and security appliances - Adaptable with current process	[17-19, 24, 74, 91, 95, 116, 125]
Cost	- Low computation cost - Cost of effective SOAR platform for mixed environments - Additional resources - Costs of ownership	[18, 19, 23, 71, 74, 84, 122]

2.4 KEY COMPONENTS OF SECURITY ORCHESTRATION

Organizations and SOCs must consider the key components of SOAR platforms before adopting them. We have identified several core components of a SOAR platform. Most of the reviewed studies have a combination of these components that we have categorized into three classes: unification, orchestration and an automation unit. This classification is based on the functionalities discussed in section 2.3.2. We have considered the external security tools as another key component of a SOAR platform because most of the SOAR vendors consider that an organization already owns some security tools and uses the capabilities of the existing security tools. Figure 2.7 presents the details of the classification of the core components of SOAR platforms.

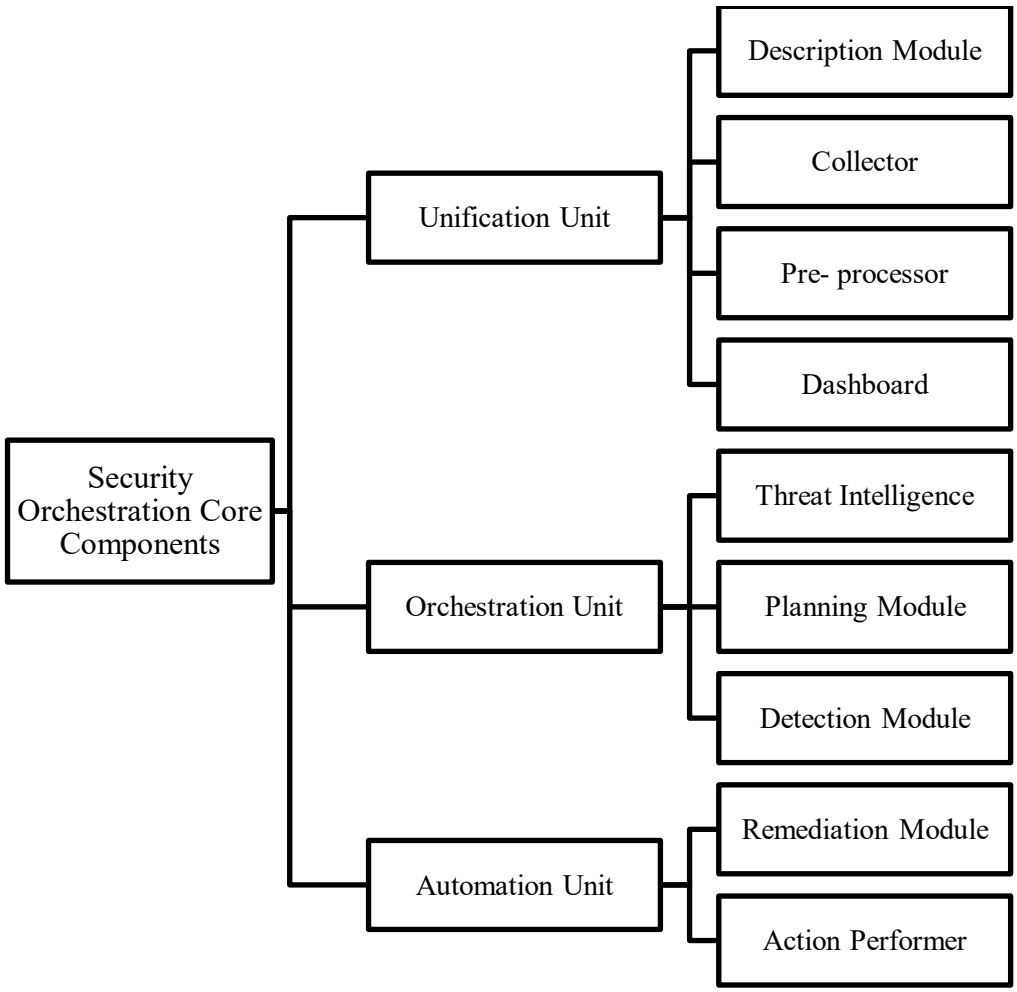


Figure 2.7 Categorization of core components of a SOAR platform

All the components proposed here comprise of learning capabilities, specific policies and storage to store the threat data. These components also store security policies

and rules associated with various organizational assets and endpoints [31]. We consider an organizations' devices that can act as an endpoint to be a server and any client personal devices, such as a laptop, mobile, personal computer, or organization-owned workstation and so on. We do not include these in the presented categorizations in Figure 2.7 for simplicity's sake.

2.4.1 Unification Unit

We have considered the components that are designed to unify the existing security tools and their activities under this category. The unification unit works as middleware or hub, as briefly discussed in section 2.3.2.1. We consider the data collector, alert pre-processor and description modules as part of the unification unit.

2.4.1.1 *Description Module*

This component refers to language and models to represent the configuration, deployment and control tasks of a SOAR platform. Luo et al. [21] have proposed a system that requires an abstract set of activities (i.e., services) to be defined for the same type of security activities. Security management is built using the API of abstract activities. Luo et al. [21] have proposed a security tool capability descriptor, which describes all types of security tools, inputs and necessary attributes. The description module requires a well-designed API to connect the existing security tools. The security activities requirements and descriptors are derived from organizational security policies and security tools, respectively. The set of interfaces are mentioned as connectors in Reference [21]. They have introduced two sets of interfaces: event connectors and command connectors. Both security and network events are received by the event connector [21]. These events come from external sources. The events connector also sends the information to the playbooks. The set of interfaces under the command connector sends a command to a security tool to modify their configurations and update operation behavior [21].

A SOAR platform needs a set of APIs to help with integration of third-party security tools and control the activities in different layers [102-104, 118, 128, 132]. Intel has proposed a bi-directional notification-based API to orchestrate virtual security in Software Defined Data Centers (SDDC) [71]. The global threat intelligence platform supports dedicated tools to provide a simplified interface to a firewall's control [72]. Swimlane also uses API to enable one-click automation [61]. As stated by Bernd [91],

one of the main purposes of ETSIs' management and orchestration group is to control the Network Function Virtualization (NFV) environment through virtualization and automation as much as possible. This work has enhanced the NFV reference architecture with a security orchestrator, the interaction of the security orchestrator with the existing NFV orchestrator and a virtual network function manager of the reference architecture. The workgroup has also included the tasks of the security orchestrator and the required interfaces with which to interact. In this work, the author has stated how security is managed through orchestration in a virtualized network environment.

The correlation module of alert correlation architecture proposed in Reference [83] uses an application interface with the reasoner for reply and request. The DXL layer proposed by McAfee for an enterprise service bus works as a connector for diverse security tools and has an extensible data exchange framework to facilitate configuration of trustworthy data representation [59]. The safeguard interface module of the enterprise-level security orchestrator provides a layer of software for a consistent interface to abstract away the changing nature of the underlying safeguard software packages.

2.4.1.2 *Collector*

Most of the reviewed SOAR platforms have collectors to collect all the necessary information from integrated security tools or devices. In several studies, network traffic and alerts are collected and pre-processed before analyzing and taking a decision [23, 63, 84, 99, 105, 125, 129, 147]. The collector collects both raw context and structured format data [103]. The orchestration server engine of the security orchestration framework presented in Reference [59] works as a collector and receives contextual data from clients. The SOAR platform discussed in Reference [23] has an OADS miner as a core component. The OADS miner works like a consultant to the overall platform, which comprises an OADS crawler. The OADS crawler is designed to gather new information from the Internet about threats, vulnerabilities, attacks, and the origin of attacks, and store them in a repository. The threat intelligence unit works as a blog scrapper that crawls through technical blogs to collect, gather and share threat intelligence data [87, 103]. A SOAR platform utilizes global threat intelligence platforms to collect external threat intelligence and to prevent data infiltration (the action of entering or gaining action) and subsequent actions performed by attackers [72].

2.4.1.3 *Pre-processor*

A pre-processor receives raw alerts from several security tools and prepares the alerts for analysis. The alert pre-processor first decides the alert adequacy and then it aggregates the alerts into clusters, based on similarity. The pre-processing of threat data involves many activities, such as sentence splitting, special content extraction, content term location, topic classification, template removing and content sanitizing [87, 99, 102, 103, 113]. Feitosa et al. [23] have used the Intrusion Detection Message Format (IDMF) standard to aggregate the alerts of several IDS tools. The aggregation has also helped to explore the distance between the times of different alerts, determine the alerts field and make a hypothesis about alerts and defense strategies. The proposed architecture for the Collaborative Intrusion Detection System (CIDS) [84] has also used IDMEF to unify alerts from multiple IDS. In Reference [21], the SOAR platform has modular physical logical attribute mapping that maps all assets' physical attributes to their corresponding logical attributes. The study in Reference [83] combines several knowledge representation languages, for example, IDMEF, TAXII, OVAL, STIX and NVD, to propose ontological conceptualization and divide the knowledge into several groups. Their proposed ontology-based event correlation architecture consists of two essential modules: one is the conversion, and the other is the correlation module. The conversion module consists of parsing reports, translation and ontology.

2.4.1.4 *Dashboard*

The dashboard category consists of tools aimed at visualizing the activities of a SOAR platform. According to Demisto Inc. [147], the dashboard will bridge the gap between the SOC and technology used to keep the organization secure. The dashboard provides an aggregated view of different scenarios, assets and metrics [32, 130, 147]. FireEye uses a centralized dashboard to facilitate advanced threat hunting [118]. In the Enterprise Level Security Orchestrator [58], the SOAR platform provides administration and an interface through a dashboard. The dashboard can be designed to provide an integrated view of an organization's overall system to help the security team understanding the security states; for example, security teams can see all the scans in progress from a single console [58, 92, 114]. The FireEye orchestration deployment service provides documents associated with cyber playbook reviews that help the orchestration operation team to

understand the playbook [118]. Most of the reviewed systems generate some form of alert reports for security teams [84, 104, 113]. Through these reports, security teams can get a high-level overview of an organizational cybersecurity tool. These reports also enable experts to identify the security state of critical assets and the affected networks or subnetworks within an organization. The reporting tools receive a recommendation from the remediation engine related to threats. Threat visualization and analysis is an important part of security orchestration. A set of papers have mentioned several web portals or public websites that provide a web interface to visualize the threats [72, 94].

2.4.2 Orchestration Unit

One of the substantial pre-requisites of a SOAR platform is to automate the control of deployment and configuration of all the security requirements. For this category, we consider all the components that are required to perform the functionalities described in section 2.3.2.2. For example, the security orchestration framework described in Reference [59] provides a security orchestration engine to receive contextual data from clients. It comprises one or more logic element(s) which are designed to work with the contextual data. Koyama et al. [72] have proposed a three steps process to cope with new sophisticated unpredictable threats: collect, judge and control. The operating principle of the proposed framework in Reference [59] has security controls, which perform context sensing and, based on the context, generate and enforce security policy decisions. A SOAR platform needs to manage several activities, as stated in References [91, 97]. The proposed SOAR platform performs central management of security activities and trust. FireEye, in their SOAR platform, have used a specialized component, called case management, for managing various cases. Thus, we categorize the orchestration unit into three components: threat intelligence, a planning module and a detection module.

2.4.2.1 *Threat Intelligence Unit*

Cyber threat intelligence can be considered as a database of evidence of existing and emerging attacks [87, 92, 103, 104, 112]. Threat intelligence consists of information related to the attack's context, adversary strategies, mechanisms, indicators of compromise, possible courses of action, tactics and techniques [32, 87, 122, 126]. Threat intelligence plays a key role in security orchestration. An organization can gain visibility about threat landscapes by using threat intelligence. It helps organizations to identify the

early signs of attacks [87]. Organizations collect and exchange threat intelligence data across several domains and stakeholders as an Indicator of Compromise (IOC) [87, 100, 101]. Example of IOC can be forensics artifacts, virus signatures, IPs/ domains of botnets and MD5 hashes of attack files. Most of the SOAR platforms consider threat intelligence as an essential element for identifying attack behavior at an early stage [23, 72, 90]. A SOAR platform with a Global Threat Intelligence Platform (GTIP) [72] incorporates proactive defense technologies, including threat intelligence.

There are several open source threat intelligence platforms that provide high accuracy and coverage [87]. Each has their own specialized techniques for data extraction. Threat intelligence data may also suffer from quality issues such as accuracy, completeness, consistency, timeliness and relevance [92]. Filtering, configuration and searching options are not available in some of the current threat intelligence tools. Xiaojing et al. have identified 45 blogs that are operated by renowned organizations and practitioners to cover major security incidents [87]. These blogs consistently publish verified IOCs that a SOAR platform can utilize for updating threat intelligence information.

2.4.2.2 *Planning Module*

We consider the cybersecurity playbook as part of a planning module that outlines the steps to respond to a security incident, including incident qualification, triage, investigation, containment, notification and post-attack analysis [65, 121, 122, 129, 131]. A playbook arranges security operations into a human-led security workflow that is a coordinated set of activities performed by various components to complete an incident response within an organization. According to an organization's policy and infrastructure, a cybersecurity playbook creates smart branching workflows and also supports the activities of the SIEM, firewall, IPS, vulnerability reporting and ticketing systems [105, 112, 147]. Incident response playbooks contain various courses of action. FireEye has proposed that an incident response playbook should be one of the key features of a SOAR platform [118]. To provide continuous proactive security, FireEye designs SOAR platforms according to an organization's requirements, integrates security tools with the SOAR platform, deploys and tests it in the organization's environment and operates it to execute an appropriate playbook against a security threat. Orchestration of

Software-defined Security Services [21] design playbooks to store the actions (i.e., operation plans) related to important security events/ security alerts. Security teams take the necessary steps, based on the actions mentioned in the playbooks [21].

Zonouz et al. in Reference [82] have proposed a consequence tree (i.e., a tree of critical assets defined by an administrator) to capture critical IT assets and organizational security requirements. Each organization has their own list and priority of critical assets. The consequence tree is built using this list of critical assets. Kamal et al. [85] consider the Incident Response Plan (IRP) to be a crucial component of collaboration engineering. The authors highlight that creating an IRP through collaboration amongst a group of experts is challenging when time is very short. The PSI policy abstraction helps an administrator to define policies in terms of what they do, rather than the details of how to implement the policies[74]. With the help of a PSI engine, an administrator can define how the traffic of a particular device should be processed and where to forward it. A SOAR platform requires proper planning to respond to incidents. Without proper planning and preparation, a SOAR platform ends up automating a poor process that might slow people down [65].

2.4.2.3 *Detection Module*

The detection module detects the anomalies and attacks around organizations, based on gathered and pre-processed data, shared insight, and knowledge of the playbook. The analyzer and the decision service unit are the two main, core components of the detection module. In the following paragraphs, we briefly describe these two components.

Analyzer*:* The analyzer receives aggregated alerts from the alerts pre-processors, correlates them, validates the assumptions and, if possible, predicts future threats and targets. The analyzer performs an automated analysis of a system, analysis of the logic unit and enriches data to boost knowledge[70, 99, 101-103, 121, 122, 129]. A set of papers have reported on correlating suspicious evidence provided by distributed security entities to identify distributed attacks [83, 84, 86, 87, 105, 113, 121]. Enterprise Level Security Orchestrator [58] can analyze alert data and detect threats. It stores all the threats data in its analysis storage. It generates a set of rules or traffic patterns as a decision table by finding the correlations across different alerts. Similarly, the PSI performs packet pre-processing and event pre-processing before analyzing the data [74].

Kenaza et al. [83] have used an ontological reasoning approach to correlate alerts. The proposed ontology-based event correlation architecture has a correlation module that works as a reasoner. Feitosa et al. [23] propose a collaborative solution to detect intrusion and anomalies by analyzing the co-creation of events and alerts among different subnetworks. It derives policy decisions, based on the contextual data that it receives from an orchestration engine [23]. The solution proposed by Reference [87] tries to find correlations among IOC to check the relationships amongst the threat data. A core part of their solution is the Analyzer. Seclius [82] tracks the interaction among files and processes to probabilistically identify dependencies among assets of an organization.

Decision Service Unit: Most of the reviewed SOAR platforms have a decision services unit that orchestrates the activities for automated decision-making [35, 117, 121, 130]. The decision service unit makes security policy decision(s) related to vulnerability and threat assessment, and assessment of security enforcement systems [99]. The decision service unit receives summarized information from the analyzer and collector about suspicious behavior and generates decisions based on that data [23].

A finite state machine (i.e., finite automata, Markov chains or stochastic regular grammars) is a popular method used in the decision process. For example, the security orchestration framework in Reference [59] uses a policy orchestration state machine to provide policy decisions to the security orchestration state machines and derive policy decisions based on the contextual data received from the security orchestration server engine. Policy decision logics are extracted from individual controls. The decision logics enable additional, ad-hoc, smart logic and intelligence analytics to be injected into the real-time policy decisions. Thus, policy decision logics capture context and drive actions staged over multiple points in space and time [59]. Similarly, Seclius [82] has constructed a dependency graph and consequence tree of existing assets to probabilistically determine the comprised assets, prioritize alerts, and provide a security state of different assets to the administrator.

Koyama et al. [72] use optimal decision-making technology and diverse threat intelligence with a variety of security sensors and appliances to choose the correct countermeasure for stopping an attacker's Internet-based actions. Utilizing a workflow engine is also a popular strategy for decision-making. For example, Rochford et al. [70]

decide on actions, based on the workflow engine. The reviewed study, SOSDSec, has generated a security service binding upon finding matching among security requirements and abstract services [21] that contains information related to assessment and its service provider. A change in the security control and module causes a change in the security service binding. Similarly, the differentiated search engine in OADS miner discussed in Reference [23] is used to generate decisions based on the queries received from end users or system tools. The decision service unit has been designed to make all the decisions related to the OADS miner, like activation, deactivation and parameter changes, and stores the configuration parameter in a file. The decision service unit also provides recommendations after analyzing information about attacks. The alert buffer of the security orchestrator proposed in Reference [58] continuously sends updates to a dashboard.

2.4.3 Automation Unit

The automation unit performs all the automated tasks, based on the decisions generated by the decision unit and analysis of the workflow. The remediation unit and actions performers help a SOAR platform to deal with the automated tasks. In the following paragraphs, the role of the remediation module and actions performers are described in detail.

2.4.3.1 *Remediation Module*

The remediation module promptly configures countermeasures and security operations, based on the decisions of the detection module to remediate threats [73, 104, 121, 125, 129]. A remediation module reported in Reference [72] performs automatic security configurations for responding against and mitigating the effects of attacks. The remediation module brings about automation in SOAR platforms, delivers significant ROI and drives downtime to remediation [125]. The OADS system proposed in Reference [23] has a central controller to implement the established sequence of actions with a process including exceptions and conditions. Enterprise level security orchestrators [58] have a remediation module that has two main elements: response storage and a remediation engine. The remediation engine has been designed to detect threat patterns. It has a learning logic module that uses machine learning algorithms. Threat data are stored in the response storage once received from the remediation engine.

Koyama et al. [90] discuss the technology to recover rapidly from the effect of cyberattacks. The proposed remediation module immediately isolates the affected area after detecting attacks, based on the information from a detection module, and provides recommendations about detected attacks for further analysis and evaluation. These actions of the proposed system are expected to reduce a security operator's burden. The SoSDSec system proposed in Reference [21] incorporates a model layer to manage the security policies and security models of an organization's assets. The security orchestrator reads and updates policies to achieve automation. An SDSec orchestrator is a key element to achieve security orchestration and automation. It works with the communication and coordination subsystem. Communication with security tools is also performed through the orchestrator. The SDSec orchestrator communicates with virtualized functions to coordinate security activities and thus minimizes management dependencies on security appliances/ tools.

2.4.3.2 *Action Performer*

A controller or action performer controls the communications and actions of a SOAR platform [121]. The security team can control a SOAR platform's various components directly through a controller [73, 104]. The action performer performs many actions such as sending an e-mail to the relevant persons, blocking an IP address, isolating a virtual machine, triggering a process to initiate a scan and running a script to perform auto-configuration [70]. Poornachandran et al. [73] refer to the data management processing system as a security and administrator console that works as a tracking station. The tools enable security teams to tackle diverse and ongoing issues [129]. The communication module can be considered a subcomponent of the controller. The job of the communication module is to work as a bridge between several components of a SOAR platform. In addition to maintaining a secure exchange of threat data and policy information, a SOAR platform requires a secure broker or DXL [59, 73, 87]. Elshoush et al. [84] consider the communication module to be a bridge between the security tools and the decision-making module. The DXL fabric of Reference [59] provides command and control functions across the entire network. Published, subscribed notifications, query responses and push notifications are different types of messages from the DXL layer.

Demisto provide DBot and ChatOps to perform intelligence automation and collaboration among security teams and security operations [35].

2.5 MOTIVATION BEHIND SECURITY ORCHESTRATION

This section reports the result of RQ2: *"What challenges is security orchestration intended to solve?"* We have identified and analyzed the challenges that promote the practice of security orchestration. Our analysis of the extracted data enables us to identify several challenges, as shown in Figure 2.8. We have classified the challenges under technical and socio-technical aspects of security orchestration.

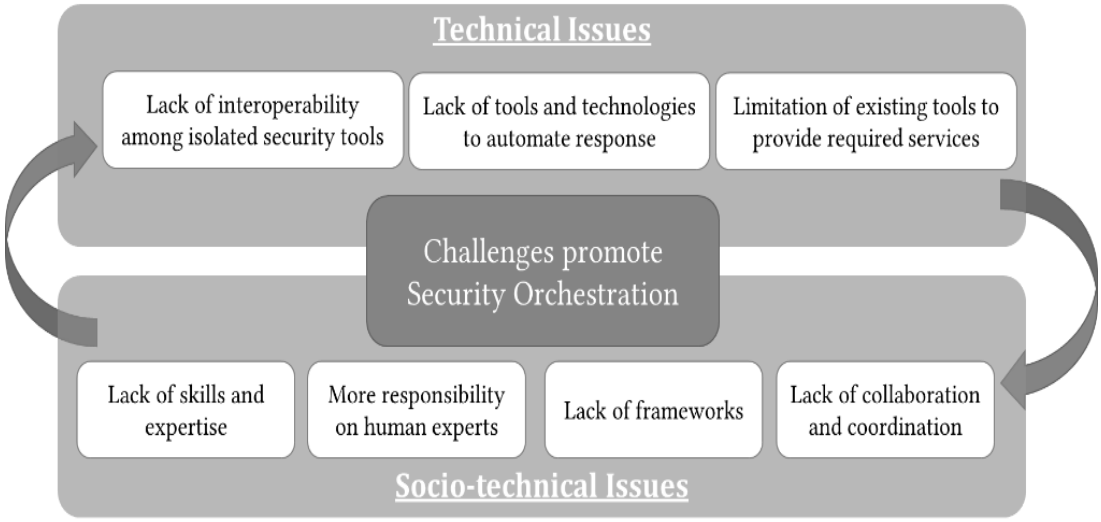


Figure 2.8 Challenges that promote security orchestration

2.5.1 Technical Challenges

Technical challenges are related to technical issues that lead to security problems such as limitations of the IDS to detect intrusions accurately and interact with other security tools, conflict among several security tools running simultaneously, and dynamic changes of security tools' behavior. The following sub-sections describe the technical challenges that a SOAR platform intends to solve.

2.5.1.1 *Lack of Interoperability Among Isolated Security Tools*

Our analysis of the reviewed papers reveals that most medium to large organizations use several security tools (e.g., IDS, Firewall and SIEM) to secure their critical data and infrastructure [18, 95, 137, 138, 151]. The main reason behind an organization installing several types of products is that different vendors provide distinct dimensions of security

tools and solutions [17, 58, 124, 145, 148]. Moreover, organizations lack a single security tool that can encompass all of the security operations needed. Isolated security tools are considered poor communicators and cannot always assume the presence of another security tool [18, 21, 23, 72, 121, 136]. Several security tools fail to guarantee the protection of an organization's infrastructure, as they work in an isolated way and focus on solving specific problems [101, 104, 122].

Several of the reviewed papers mention that it is extremely difficult for a single security tool to detect the distributed and complex behavior of cyberattacks. Moreover, security operators are usually unable to understand their organizational security state through individual security tools working separately [82]. To take incident response decisions, it is necessary to integrate and analyze the activities of different security tools, which are usually designed to work independently and are limited by their own services [72, 101, 104]. These tools have their own data representations and interpretation mechanisms. The disparate tools have inconsistent workflow [29], disconnected and non-integrated architecture [18] and a lack of standardization for data exchange between different security tools [92]. These are some of the reasons that network administrators and security experts find it difficult to appropriately configure and integrate the activities of multivendor security tools, which means there is a need for the continuous involvement of humans in the entire process of a security incident response.

The lack of interoperability among security tools results in more responsibility being placed on human experts (briefly explained in section 2.5.2) and leads to redundant, complex and inefficient incident response processes. Existing security management and risk assessment solutions are not designed to collaborate [116]. These solutions do not consider several aspects that affect the evaluation criteria of the threats and vulnerabilities, thus making the security procedures incomplete. As a result, with generic security policies, security management becomes inefficient [116].

2.5.1.2 *Lack of Tools to Automate a Proactive Response*

Our review has revealed that there is a lack of tools to automate key security activities, such as threat intelligence collection and update, alert validation, task investigation, response and resolution [70, 121, 137, 141, 145, 147]. Organizations need tools to automate the repetitive manual tasks. FireEye mentions that security teams spend 95%

of their time on the manual execution of repeatable tasks [29]. AT&T's cyber securities insightreport has revealed 90% of their reported cyberattacks were from known vulnerabilities [145]. Whilst security defenders need to update new threat intelligence quickly, they usually fail to instantly update the threat intelligence [59, 87], promptly update software patches to remove vulnerabilities [105], keep every security tool up to date [137] and enforce policies as soon as they are agreed upon [74]. For a large network, it is time-consuming to update hosts from different vendors that leave the system open for intruders [105, 121, 122]. Ntouskas et al. [116] propose that there is a lack of automated collaborative tools to embed security standards, methodologies, tools and guidelines to train a security management team as one of the key reasons organizations lag behind in fulfilling their security needs. Fujitsu emphasize that an efficient SOC requires automation of the process of the threat defense life-cycle to help free up security analysts' (i.e., the security team's) time and keep the system up to date [114].

2.5.1.3 *Limitations of Existing Security Tools to Provide Required Services*

Several of the reviewed studies have mentioned that the existing security tools are unable to give full protection to organizations' infrastructure [70, 132, 137, 146, 148]. According to Verizon's 2017 Data Breach Investigation Report [49], 43% of data breaches utilize phishing techniques and it is clear that trying to prevent every attack is an impossible task. A single, standalone detection engine also fails to provide complete visibility of the network infrastructure to the security team. In most cases, the detection system generates a large number of false alerts that require extensive analysis [18, 19, 29, 60, 121]. Security teams are overwhelmed with alerts and spend more time investigating and validating false and repetitive alerts than solving real attacks. In 2015, Hewlett-Packard reported 48% of their recorded cyberattacks were from known vulnerabilities that are five or four years old [145]. Organizations need tools that can learn from experts' behavior. There is a lack of a platform whereby security teams can easily integrate security tools, network infrastructure and gather complete visibility of their cybersecurity tools [70, 124, 128]. Weilinger et al. [107] have reported that IT security tools used by security practitioners fail to address the complexity of their interactions. According to Demisto [147], an inappropriate interface between technology and personnel is the reason for security teams being ineffective and inefficient. The IT

security tools provide insufficient support for collaboration, coordination and cooperation among security practitioners and stakeholders.

2.5.2 Socio-Technical Challenges

Socio-technical challenges are related to the organizational process, policies and rules with respect to cybersecurity. Socio-technical aspects of security in an organization include matters involving business processes, skills, resource management, policies, law enforcement and interaction of people with the technical system. Many of the challenges faced by the security community are socio-technical rather than technical. Socio-technical challenges are difficult to project as they involve interactions between individuals, groups and technical systems. Our analysis of the extracted data indicates some of the key socio-technical challenges that organizations face while handling a security incident. These challenges work as the primary drivers of security orchestration.

2.5.2.1 *More Responsibility and Workload on Human Experts*

Security teams are entrusted with several types of responsibilities, which include analyzing and dealing with sophisticated attacks [90], manual consultations and writing custom codes to validate alerts through threat intelligence [19, 114, 142], manual extraction of key attributes from threat intelligence data and linking them with relevant data [103], evaluating alerts, correlating data, coordinating the appropriate responses [17] and investigating results [105, 148]. Several papers [18, 19, 21, 72, 90] report that the response toward a security incident highly depends on the manual activities performed by security teams. A security team needs to combine several security tools [107], update threat intelligence, is involved in multiple administrative systems, including multiple control tools [72], analyzing data from new tools [136] and dealing with the inter-component interaction of a modern complex system [63, 95] to perform their tasks. The manual steps are usually the main reason for longer incident response times [17, 124, 132, 136]. A delay in the security incident analysis happens when security teams need to continuously shift between multiple disparate security tools to manage the different pieces of information generated by these security tools [29, 65, 137]. Fujitsu's SOC considers manual consultation as one of the most time-consuming steps in the incident response process [114].

Hexadite report [19] that it takes around 45 days for an organization to resolve cyberattacks due to manual responses to incidents. According to a set of studies [87, 103, 138, 147], security teams find difficulties with manually extracting features from huge volumes of threat intelligence data. Manual configuration, integration of several security tools, implementation and updates are associated with misconfiguration, erroneous responses and policy enforcement [92, 95, 132, 138]. Moreover, security teams face difficulty with dealing manually with the interaction of inter-components of modern complex systems [95]. Several papers [82, 83, 142] indicate that dealing manually with thousands of alerts to choose the right course of actions results in missing critical attack information.

2.5.2.2 *Lack of Skills and Expertise*

Security practitioners report [17-19, 29, 65, 136, 147] that the lack of skilled security teams is one of the major reasons for organizational failures to deal with security breaches. Large organizations are spending billions of dollars on buying and deploying several types of security tools [137, 144] that need up-to-date knowledge and expertise in different aspects of cyberattacks and countermeasures. Organizations face difficulties with finding and retaining security teams with the required expertise [65]. Security teams require a decade to acquire the expertise to fight against sophisticated cyberattacks [132]. The 2019 $(ISC)^2$ cybersecurity workforce study estimates that the current cybersecurity workforce comprises 2.8 million professionals and estimates that 4.07 million professionals will be needed to close the skills gap in the cybersecurity domain [25]. The demand for skilled or experienced cybersecurity professionals is one of the biggest challenges faced by the cybersecurity industry, [12]. According to CyberSeek, cybersecurity data tools [155], 40,000 jobs for information security analysts remain empty each year in the USA, whilst organizations struggle to fill 200,000 other security-related jobs. Organizations have few security experts to deal with the thousands of incidents that they receive each day [18, 65]. Security teams need to have an overall knowledge about organizational security policy, network infrastructure and security tools. One study reports that organizations are continuously shifting towards modern technology paradigms (e.g., cloud computing, mobile computing and IoT) that lead to an

expanded cyberattack surface and thus need security knowledge, incident response skills and resources for each technology initiative [141].

2.5.2.3 *Lack of Regulation and Policy Framework*

One of the major challenges with organizational security tools is the lack of a fully developed framework for conducting IRP [85], performing coordination and collaboration among incident responses [100] and seamless implementation and deployment of policies [59]. Some of the challenges mentioned in the reviewed papers include failure to provide a clear definition of unwanted traffic and network behavior [23], significant difficulties in providing clear guidelines to deal with new security mechanisms [21, 107, 112], failure in providing appropriate training for security management [116], a lack of guidelines for conducting incident response planning [85] and severe challenges in enforcing and managing security policies. All these types of challenges result in security teams failing to take proactive decisions against cyberattacks.

2.5.2.4 *Lack of Coordination and Collaboration among Stakeholders and Security Teams*

Coordination and collaboration among security teams are important for analyzing complex threat behaviors. Most security teams lack collaborative processes for information sharing. Several papers [62, 85, 86, 88, 94, 107, 117] highlight the requirements for having combined knowledge and experience from several domain experts due to the complexity of the network flow and log data analysis. Most incident response teams follow no collaborative process while planning how to respond to a particular incident, which results in poor strategies planning [85]. Several papers [72, 98, 117] have revealed that stakeholders from different organizations are unwilling to share threat intelligence with each other. Jeong et al. [117] report organizations' fear of losing their reputation is one of the reasons for their unwillingness to share their security circumstances with other organizations. Zhao et al. [98] have discussed that many state and federal governments have developed threat information sharing services that are limited to sharing threat intelligence with central government. External organizations do not benefit from this kind of threat information sharing. Still, there is rarely collaboration among different organizations working in the same domain [72].

Table 2.6 presents the mapping of the benefits that organizations obtain from the functionality discussed in section 2.3.2, which aims to solve the challenges discussed above in this section.

Table 2.6 Mapping summary of key activities performed by a SOAR platform with benefits of SOC

Key Functions	Activity Performed	Benefit to Organizations	Articles
Unify security tools	• Unifies disparate security tools and processes • Integrates enterprise security architecture • Connects detection, networks and endpoint security tools • Performs coordination among security tools' activities • Unifies intelligence according to vulnerability • Removes the operational silos.	• Efficient and effective incident handling processes • Frees up experts' time • Minimizes the overall complexity of the incident response process • Enhances organizational protection and defense systems • Experts operate disparate tools as a unified system	[17-19, 21, 29, 58, 59, 61, 65, 72, 83, 92, 101, 124, 128, 130, 136, 143-145, 148, 154]
Determine Endpoint for human investigation	• Works as a helping hand for security experts • Informs and educates security analysts about threat behaviors • Decides when human insight is needed • Defines source of information to help experts solve problems	• Keeps analyst focused on threats that demand their ability • Reduces human error • Faster decisions • Reduces burden on security operators	[17, 19, 24, 59, 61, 72, 90, 120, 128, 137, 144]
Share contextual insight (via platform)	• Gathers threat intelligence from various external sources • Extracts key features from threat intelligence data • Provides contextual insight related to alerts or attacks to the security analyst • Provides real-time visibility • Context-aware frameworks • Gathers an overview of what is happening in various subnetworks within the organization	• Experts get the insight of several security controls activities • Organizations share contextual device data with third-party systems • Reduces and mitigates risk exposure • Faster decisions	[18, 24, 29, 35, 59, 72, 82, 85, 86, 88, 93, 94, 102, 136, 144, 145]

Key Functions	Activity Performed	Benefit to Organizations	Articles
Translate complex processes into a streamlined workflow	• Allow experts to simplify high-quality workflow integration • Coordinates the flow of data and tasks by integrating tools and processes into automated workflows • Enables powerful machines to undertake machine automation • Offers data aggregation to provide in-depth awareness about the environment	• Reduces human error • Improves staff capability for incident responses • Provides standardized processes • Reduces reliance on human expertise • Simplifies security management tasks	[17, 19, 21, 29, 71, 74, 82, 136, 137, 139-141, 143, 148, 154]
Provide deployment model	• Resolves incidents in minutes to determine appropriate and effective course of actions • Determines a proactive response to threats • Initiates additional investigations, based on the level of the attack's complexity	• Accelerates response • Mitigates conflict installation • Reduces conflict configuration • Minimizes the effect of attacks on services	[71, 74, 86-88, 90, 112, 118, 146]
Determine appropriate course of actions	• Simplifies threat responses through integration and automation • Decides on additional investigations • Limits execution access and privileges to workflows alone • Dynamically inserts security functions into the workgroup, based on policies • Engages the security tools to perform complete monitoring of the endpoint and correlates their activities • Performs coordination and collaboration among different anomaly detection techniques to detect and evaluate threats and choose right actions	• Maintains process consistency across security programs • Reduces manual investigation errors • Maintains effective communication and strong collaboration among cyber security teams • Simplifies and accelerates alerts investigations • Accelerates Return of Investment (ROI)	[17-19, 21, 29, 58, 59, 71, 72, 82, 90, 112, 117, 118, 139, 142, 144, 148]

Key Functions	Activity Performed	Benefit to Organizations	Articles
Automate repetitive and manual tasks	• Automates repeatable tasks Automates deployment of security functions over the network infrastructure • Repeatable, executable planning processes • Eliminates the need for continuous vendor assessment	• Optimizes security teams' capabilities • Reduces cost • Minimizes mistake-prone manual processes • Accelerates responses	[17, 29, 71, 85, 87, 102, 144, 146, 147, 154]
Automate policy enforcement	• Automates policy enforcement and configuration at runtime • Real-time policy enforcement	• Minimizes mistake prone configurations • Reduces conflict configuration.	[21, 59, 71, 73, 74, 91, 98, 99, 101, 104, 141, 144]

2.6 TAXONOMY OF SECURITY ORCHESTRATION

In this section, we have summarized the results that answer to RQ3: *"What types of solutions have been proposed to adopt security orchestration?*" We have highlighted the key techniques, tools and strategies used by practitioners and researchers in the realization of security orchestration. Most of the reviewed studies propose platform-based architectures as a strategy for incorporating security tools to support their unification, orchestration and automation [18, 19, 118]. McAfee focuses on four engineering approaches to automate the entire threat defense life-cycle: partnership centric, platform-based approaches, reinvented experiences and cloud-centric approaches [18]. All four approaches are integrated into a single platform to take the benefits of each. We consider the platform-based approach as the core engineering strategy. A SOAR platform is designed to automate various activities in the threat defense life cycle. This review has enabled us to propose a taxonomy of security orchestration to support a systematic comparison and analysis of the existing security orchestration solutions, as depicted in Figure 2.9. The proposed taxonomy consists of several dimensions and sub-dimensions for classifying security orchestration techniques.

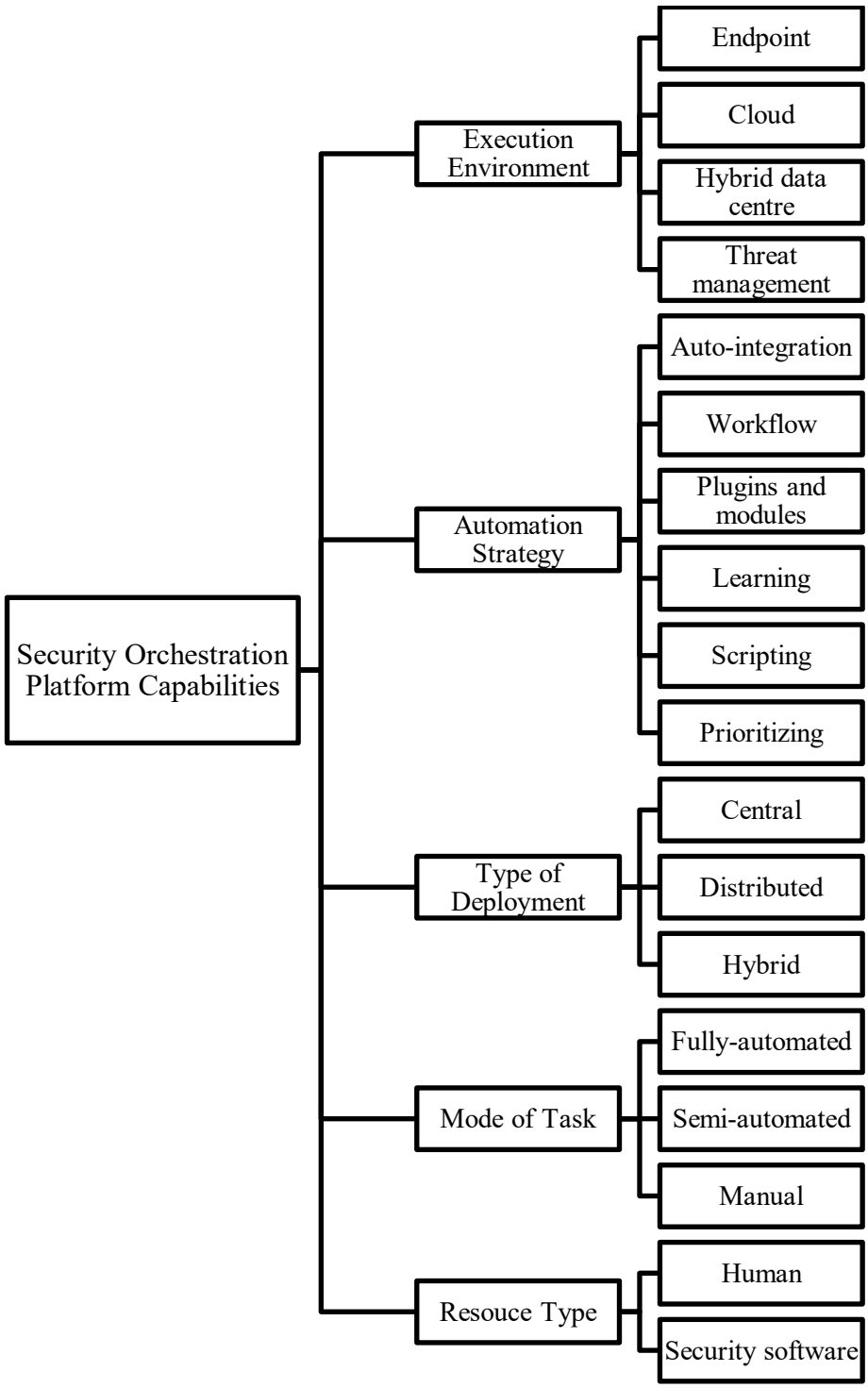


Figure 2.9 A Taxonomy of an Orchestration Platform

2.6.1 Execution Environment

To help speedy organizational responses to security incidents with fewer resources, a SOAR platform's execution environment can be supported by four types of technological solutions that are expected to work together to solve security issues and challenges. For

example, a combination of cloud-delivered data security tools and endpoint security for Infrastructure as a service, and multiple vendors with multi-tenancy features need to be considered in a virtual context. Another example is that ETSI has proposed a security orchestrator for a hybrid network consisting of a physical network and a virtualized network [91]. In the following subsection, we discuss the four execution environments for security orchestration solutions.

2.6.1.1 *Endpoint*

Most of the organizations have several siloed security tools in their endpoints. Installing several security tools in each endpoint and managing the endpoint in a large IT infrastructure is becoming challenging and inefficient [127]. An organization needs to monitor, assess and control all the endpoint devices connected within the organizations' network to provide end to end threat protection [124, 127, 130]. McAfee has considered an endpoint security architecture from which an organization can expect agent consolidation. In this review, we consider any autonomous entity or software program that can perform actions as an agent. A SOAR platform can deliver consolidation at the endpoint, which can even be the entire portfolio, depending on time [18]. A SOAR platform agent can reside in various endpoints' storage (RAM, HDD or SSD) [73].

HEXADITE has proposed a SOAR platform, AIRS, that helps an enterprise to connect detection, networks and endpoint security tools. Though the proposed platform seems to be agentless, it uses a non-persistent agent that injects a dissolvable probe into endpoints during investigations [19]. Similar to Hexadite, Demisto [148] proposes an architecture that consists of dissolve agents for data collection from endpoints. The workflow tool CounterACT proposed by ForeScout uses multiple agentless discovery methods and integration techniques. CounterACT employs a combination of active and passive discovery methods to classify organizational devices based on a network [144]. Without installing any software agent or enrolling any management unit to a device, it first connects the device to the network. This reduces the overhead of an administrator to check each device and manually assign policies to each endpoint. The resilience engine proposed by NTT controls multiple devices at appropriate points, according to the type of attacks, to isolate the affected regions [90].

The security orchestration framework proposed in Reference [59] also supports distributed endpoints with DXL over an enterprise network. DXL is built on top of Enterprise Service Bus (ESB) technology and provides an abstraction layer between different types of connected endpoint devices. Through a SOAR platform, organizations can provide constant protection, irrespective of where an endpoint device is located.

2.6.1.2 *Private and Public Cloud Computing*

Cloud computing and its related technologies have created the need for a new generation of security technologies. A SOAR platform can be built as a single integrated solution to provide cloud-delivered data security [18, 127]. The motive for a cloud platform is to build software as a service, with the required levels of performance and availability. An organization can easily integrate their security tools (i.e., service) into a cloud [127]. Example of such services includes web protection, sandboxing, security brokers, data loss prevention and encryption. A cloud security platform continues to support next-generation platforms that are built beyond VMware and Amazon web services to add Azure, OpenStack, Docker containers and emerging services. McAfee has proposed McAfee cloud ePO software to support consolidated management across their cloud management technology [18]. The work proposed in Reference [21] is designed to deal with heterogeneous cloud environments and automated security operations in a Software Defined Infrastructure (SDI) environment. The proposed solutions handle VM movement over dynamic infrastructures and provide transparent security management facilities. The resilience security technology for rapid recovery from cyberattacks also works for network services in cloud environments [90]. The enterprise-level security orchestrator installs a mirror of the SOAR platform in a cloud [58]. The authorshave proposed a security orchestration engine for both the server and the client and the SOAR platform can be used in public, private or even external cloud facilities [58]. Using the cloud as an execution environment helps an organization to have scalable, flexible and adaptable infrastructure.

2.6.1.3 *Hybrid Data Centers*

The evolvement of a data centre to SDDC has created new security-related challenges for organizations. Additionally, the increasing trend of distributing more workload on data centers and public clouds has also increased security challenges. The SOAR

platform and resilience engine proposed by NTT are mainly designed for a data centre [72, 90]. McAfee aims to build an integrated platform to deliver visibility and security to a cloud-enabled data centre [18]. McAfee's SOAR platform includes global load balancing infrastructure with a Content Delivery Network (CDN)/ peering data centre [18]. Intel has introduced the open security controller, a SOAR platform to orchestrate virtual security in SDDC [71]. The purpose of this platform is to make security management visible, effective, agile and scalable by providing automated, dynamic and synchronized security services for software-defined infrastructure. It provides seamless brokering services between SDN and VNF. It is optimized for an OpenStack and VMware cloud environment.

A SOAR platform gives visibility across the network and server tiers and public/private cloud data centers [127]. Dynamic micro-segmentation is performed for private clouds and workload auto-discovery is undertaken for public clouds [18, 71]. The concept of micro-segmentation restricts access and tailors security configurations. This gives better threat protection and faster remediation than siloed approaches. A SOAR platform helps security administrators to span their security model from an organizational data centre [127].

2.6.1.4 *Threat Management*

A SOC suffers from a large volume of data, events and Indicators of Compromises (IOC) to prioritize true attacks in process or in the golden hour of post-breach [84, 86, 87]. A SOAR platform helps to analyze the security threat data by providing security analysis, threat and vulnerability management, attack detection, attack investigation and streamlines incident responses [18, 72, 90, 98, 127, 130]. Threat management includes prioritizing threats in progress and also in the golden hour post breaches [18]. It helps security analytics to continue advanced data management, risk assessment, correlation and deal with both the volume of security data and increasing sophistication of analysis [23, 92, 93, 98]. It automatically investigates attacks during and after a breach. It also provides both on-premise and cloud-based analysis. Threat Connect has proposed an intelligence-driven platform to manage both internal and external threat data and turn them into actionable threat intelligence [122]. A SOAR platform provides a central place for data aggregation, analysis and enrichment of security threat data. It allows security

teams to provide technologies like attack reconstruction that help an organization to identify and respond at a full attack level, not just at an event or malware level. An organization can use a SOAR platform to centrally manage the threat and automate the entire life cycle of threat defense.

2.6.2 Automation Strategy

Our review has revealed that a SOAR platform uses a combination of several types of automation strategies. Orchestration of different automated steps is needed for effective incident responses that suit all types of organizational activities, such as integration, aggregation of data, auto investigation or analysis, finding proper courses of action and deciding on remediation processes. The security automation realization approaches concern specific methods/tools. HEXADITE has highlighted five distinct approaches to security automation adopted by current vendors: workflow tools, orchestration tools, scripting tools, prioritization tools and intelligence security automation [139]. From the analysis of the reviewed literature, we consider intelligence security automation to be a SOAR platform that includes some of the available automation tools to orchestrate and automate the incident response process. Demisto has mentioned automation and human tasks need to be interwoven and worked together in a seamless fashion to achieve a desirable goal [148]. We have outlined the automation strategies that are used by practitioners in various organizations.

2.6.2.1 *Auto Integration*

We have placed the connecting or integration tools that are used to automatically connect existing security tools through APIs to streamline an incident response process under this category. Whilst some practitioners have mentioned the connecting tools as orchestration tools [139], this is not the only purpose of using a SOAR platform. A SOAR platform needs to have tools to automatically connect and integrate a full stack of security tools [17, 118, 139]. Several reviewed papers have also proposed the connection of organizational hardware, software and control unit into a SOAR platform [146, 154]. This work as a layer of connective fabric that makes the security tools work together.

The integration tools help isolated security tools to interoperate with each other. Organizations can easily buy a new point of product, as the integration tools automatically connect and integrate new systems into existing ones, and make the

necessary changes in a system. An organization can dynamically insert security functions into any workgroup, based on their policies [71]. The ControlFabric interface by ForeScout uses an open standard based API to perform bi-directional integration [136]. Building a fully autonomous, integrated set of tools is very difficult due to the heterogeneous nature of multivendor security tools. The work reported in [59] enables the integration of third party software. Security management software can connect to the orchestration framework by connecting to DXL. It provides all the command control functions across the entire network. The DXL also provides API embedded with McAfee agents. However, if a security orchestration process is not well-defined [139], there are few benefits in simply connecting the existing security tools. A SOAR platform must have a well-designed framework and workflow to perform the required actions.

2.6.2.2 *Workflow*

Organizations usually use workflow tools to streamline an incident response flow and communication. Workflow tools are depicted as a solution to gather and enhance alerts that automatically send instructions to analysts, auditors and other security tools [17, 71, 125, 136, 139, 148]. Some workflow tools provide a standard framework, specifying user roles and types of actions that are needed to perform during certain types of alerts [70, 121, 122, 125, 130, 139]. FireEye has built a security orchestrator to design a workflow [118]. A SOAR platform can help organizations to organize incident response flows more efficiently with a built-in ticketing system. According to HEXADITE and KOMAND, the workflow tools automate data gathering and communication processes, leaving the investigation and remediation actions for the security team [17, 139]. The security team creates a sequence of automated tasks to perform the tasks in a logical sequence with a chained data flow [61, 63, 121, 122, 148]. Demisto mentions designing the workflow for automation of playbooks to weave human analysts into the middle of these workflows and playbooks. Some practitioners have designed the workflow in such a way that it will also trigger investigation and remediation actions [71, 121]. Some workflow tools, like the one proposed by McAfee, drive cross endpoint workflows and are built natively into an endpoint security architecture [18].

ForeScout has built CounterACT, which uses both a rule engine and a workflow engine to automate the workflow for instant decision-making to deal with security

incidents [136]. This helps an organization to automate the security process across mobility management and endpoint platforms. Invotas Inc. has designed a multistage workflow, which includes the workflow of automatically connecting different security tools [154]. The majority of SOAR practitioners have built workflow based on use case scenarios [17, 19, 118, 125, 136, 147]. An organization can define its strategies about how to respond to certain security events. The cybersecurity playbooks keep a record of this in the form of a workflow rule that an orchestrator uses to autonomously control attacks [21, 65, 121, 125]. Workflow tools do not enable the integration of heterogeneous inter-organizational information and security tools.

2.6.2.3 *Scripting*

Scripting tools perform actions based on custom code written by security teams, who use the scripting tools to configure existing playbooks, security tools and policies. An organization requires skilled developers to consistently write and maintain code by performing in-depth investigations [65, 125, 139]. Scripting tools can be considered to be an execution engine that executes the script or configurable code. The SOAR platform proposed in [72, 90] uses scenario-based autonomous control of multiple virtual appliances to implement security measures. A security team can implement the measures regardless of their skill level. Scripting capabilities also include writing custom workflow and integration codes [125]. An organization with the resources to investigate and remediate threats can use scripting tools to perform automation. An organization needs to have both budgets and resources for using scripting tools. Defining new policies and designing scripts according to organizations' budgets and resources can be considered under this category. The policy maker explicitly writes code to reconfigure some parts of a network. The enterprise-level security orchestration has an orchestrator routine to make calls to safeguard software packages via automated interfaces provided by safeguard interface modules [58].

2.6.2.4 *Prioritization*

Prioritization tools help security teams to decide on critical security alerts. These types of tools normally assign a score to alerts to reflect more critical and urgent alerts and prioritize security events [63, 83, 105, 114, 122, 139]. Most organizations have some sort of prioritization tools within their detection system that automatically investigate and

correlate alerts to reduce false assumptions and give experts a list of critical alerts, which are produced by the organizational detection system. Major data breaches show that, in most cases, organizational security teams have missed critical alerts. SIEM [83, 121, 147] is a popular prioritization tool that collects and aggregates alerts from different security tools and prioritizes true alerts and discards false alerts. Tayeb et al. [83] propose ontological reasoning approaches to reduce the false alerts by correlating alerts. Some of the SOAR platforms use the existing SIEM technology to prioritize the alerts.

2.6.2.5 *Learning*

A set of studies have used Artificial Intelligence (AI) techniques and game theory models to make security tools intelligent [63, 84, 88, 105, 112, 116]. McAfee has proposed an expansion of the SOAR platforms' capabilities by including behavioral security; for example, pre-execution, post-execution, machine learning and more [18]. Several of the reviewed studies [18, 20, 63] use machine learning based solutions to analyze security behaviors. The AIRS platform, a SOAR platform proposed by HEXADITE, uses AI to automate the activities of several security tools [142]. HEXADITE has proposed the use of AI as a critical capability for automated security technology [142]. Without automatically learning, it is not possible for any SOAR platform to predict uncertainties. Demisto has introduced ChatBot, a learning tool which combines intelligent automation with collaborative, human, social learning and experience [147]. For certain threat behaviors, defining rules and designing workflows works well. With a world full of uncertainty, a SOAR platform must be able to learn from a security team's behavior and threat data. A combination of AI techniques (such as Machine Learning and Genetic Algorithms) has been used in SOAR platforms' automated learning modules. Seclius [82] uses a set of instruments to learn from the dependency of systems' assets and captures information flows between files and processes. The authors have also developed an algorithm to use with the set of instruments [82]. As a result, an administrator does not need to define low-level input.

2.6.2.6 *Plugin and Module*

For this category, we have classified small programs or software that organizations that can independently select and install, based on the required configuration. A SOAR platform can integrate plugins to automate various activities and create workflow [104,

125]. Siemplify has introduced a plugin framework for security orchestration that makes security tools accessible and easy to integrate into incident response workflows and automation [125]. Komand has introduced several plugins to include in an organization's environment [17]. Each plugin has a set of tasks for a specific set of activities. FireEye orchestrator also uses predefined plugins to perform workflow integration [118]. This makes a security team more agile. A module-based automation strategy helps an organization to choose an integration module based on organizational infrastructure, policies and configuration. The ForeScout SOAR platform supports more than 70 third party solutions to automate various activities of security tools [136]. ForeScouts' open integration module allows customers, system integrators and third-party product vendors to integrate their products with ForeScout's CounterACT and communicate with each other. Both modules and plugins add specific features to a SOAR platform, which is why we have placed them under a single category.

2.6.3 Deployment Model

A SOAR platform includes several types of components. It might have several structures to manage its components and activities. Some of them form a distributed structure, while others become part of a central management site for a large-scale deployment [65]. ForeScout mentions three deployment models for their proposed SOAR platform: centralized, distributed and hybrid deployment architecture [144].

2.6.3.1 *Centralized Deployment*

In a centralized deployment architecture, an organization has a centralized orchestration manager to communicate, manage and deploy policies to multiple orchestration appliances in a data centre or major sites. Several papers [61, 104, 122, 124, 125, 128, 151] have indicated that a centralized SOAR platform is needed to provide security teams with a better understanding of the state of security throughout an organization for faster and more efficient incident response actions. For example, NetSec has proposed a central management approach for large-scale deployment of SOAR platforms [104]; whereas ThreatConnect [122] has proposed a central intelligence-driven platform to manage threat data in a single place. A centralized management configuration is necessary for optimal security enforcement. In this type of deployment, the appliances need IP connectivity to remote sites in order to manage devices and other endpoints located there.

Traffic from the remote location is sent to a centralized SOAR platform via a predefined interface for monitoring and assessment. A SOAR platform can monitor the activities of the user directory, DNS and DHCP to detect threats or potential rogue activity and initial remediation [144]. An organization manager contains the database of endpoints (active or passive) from the appliances it manages [103].

2.6.3.2 *Distributed Deployment*

An organization can use a decentralized deployment model for a mixture of security orchestration components located in both a central facility and various remote sites [98]. A controller manages and controls the activities, provides policies to orchestration service consumers, and maintains a database of active and inactive endpoints. A distributed deployment enables the use of virtual firewalls, virtual security services, browser redirection and endpoint authentication to a server when a local SOAR platform is at that site. A distributed organization, large data centre, cloud platform and large IT infrastructure require distributed deployment of security policies and protocols that can be achieved by distributed SOAR components over multiple endpoints.

Incorporating distributed security analysis and monitoring allows an organization to deliver tighter security policies and better protection against emerging cyber threats [127]. ForeScout [144] has introduced SOAR organizational controller functions that are the central notification points, where the communication occurs via email or syslog and bi-directional SIEM services via CEF or LEEF messaging to perform endpoint actions and to notify systems about each endpoint's status. Radwane et al. [86] have proposed a distributed collaborative architecture to perform cooperation and placement of defense entities on organizational systems to defend against DDoS attacks. They have utilized the concept of a distributed hash table and overlay network to perform the distribution and placement of security tools. Fung et al. [88] have used a Chord overlay network to implement the protocol of their distributed system.

2.6.3.3 *Hybrid Deployment*

The hybrid deployment model uses a mixture of SOAR platform components in a central location and at remote sites. A SOAR controller maintains a database of the infrastructure and issues policies for the applications and components. Chen et al. propose a centralized controller for managing the distribution of applications and components [103]. A hybrid

deployment implementation supports virtual firewalls, browser redirection and authentication verification of an endpoint to a server when a local SOAR application is deployed at that site [111]. Elshoush et al. [44] highlight several hybrid deployment models for a collaborative intrusion detection system. Their proposed architecture can also be considered a hybrid architecture. The security orchestration framework reported in Reference [59] has performed distributed sensing over both the server and clients. The proposed system performs a centralized aggregation of data and enforcement policies on both the server and clients. In Reference [59], a centralized server has the visibility to see the entire context and communication layer DXL, which is highly scalable, based on an elastic architecture that supports multiple deployment options. Multiple security tools can be connected and deployed at diverse locations with distinct types of capability and visibility where a data exchange layer (DXL) provides a fabric to help them operate as a unified system (super-control).

2.6.4 Mode of Task

A SOAR platform generates remediation actions that are both automated and semi-automated [80, 104]. Some actions need human involvement, depending on organizational policies and rules. SOAR platforms require a combination of machine-driven and human-led processes and workflows to optimize security operations [90, 115]. Actions can be triggered either by a security team or when a new artifact is added to an incident [115]. A SOAR platform works as an intelligence assistant for security teams, who should conduct automation selectively, based on their resources and needs. The incident response can be fully automated or semi-automated [84, 90], depending on the nature of the tasks to be carried out. For example, a task such as notifying stakeholders, assigning incidents and enriching data with context can be automated safely, but the actual containment of a data breach and analysis of unknown threats frequently requires humans to be in the loop [84]. The online evaluation framework proposed in Reference [42] does not respond automatically to an attack. Instead, it is designed to help security administrators by providing situational awareness capability. Xiaojing et al. [47] have made feature extraction and analysis of threat intelligence data fully automated. They propose a fully automated cyber threat intelligence gathering solution to lessen the manual task of threat intelligence analysis.

A SOAR platform integrated with a global threat intelligence platform provides both fully automated and semi-automated tasks [90] that help to automatically classify the detected cybersecurity attack, and investigate whether or not the available countermeasures are possible. The system also investigates the possibilities of automated responses (i.e., automatic generation and notification of response recommendations that guide the decisions of a security team). An administrator specifies the service requirements based on security tools and needs in the system proposed in Reference [31]. A SOAR platform allows security teams to choose their level of security and types of responses. A SOC can take control of an organizations' cybersecurity tools to combine various security tools and applications.

2.6.5 Resource Type

Analysis of the reviewed material reveals that the functionality and performance of a SOAR platform depend on the human expertise and security tools of an organization. We consider organizations' security tools and human resources as the two most important resources of a SOAR platform. Building a SOAR platform on top of a clumsy list of security tools that is supported by an unskilled security team will bring few benefits to organizations.

2.6.5.1 *Security Tool Resource*

In this category, we consider the existing security tools provided by third party vendors or owned by an organization. Most SOAR platforms assume that organizations already own multivendor security tools. Several papers have mentioned a range of security tools while designing a SOAR platform for small to the medium organizations [19, 23, 29, 59, 71, 99, 105, 116, 136]. Some key types of security tools used by organizations are SIEM, forensics tools, signature-based control tools, firewalls, IDS/IPS, anti-malware, antivirus, perimeter security tools, ticketing solutions, traffic inspection tools, compliance tools and vulnerability scanners [29, 83, 148]. McAfee classifies the existing security tools into attack detection and attack investigation [18]. Komand mentions IDSs, firewalls, ticketing tools and team communication tools as the minimum number of tools an organization must have to build a SOAR platform [17]. To further enhance performance, Komand considers threat intelligence, malware analysis tools and forensics tools as the next layer of security tools [17]. Komand has also considered some additional

security tools, such as applications for vulnerability scanning, phishing investigations, threat hunting, monitoring tools and malware protection tools [17].

Most of the reviewed literature has considered anomaly detectors to analyze traffic for identification of potential attacks and abnormal traffic. Kenaza et al. [83] have performed cooperation among IDS, network scanners and vulnerability scanners to reduce alerts volumes. Feitosa et al. [23] mention two types of anomaly detectors: hardware and software based. They mention several hardware tools to capture network traffic [23]. These security tools can also inspect network traffic in real time. Several tools, techniques and systems are used as software-based anomaly detectors, such as IDS (Snort, BrO, & Prelude), Honeypots (Honeyd, & Nepenthes) and open software prototypes [21, 23]. The security tools give alerts to an orchestrator and receive script commands from the orchestrator to respond to security incidents.

2.6.5.2 *Human Resources*

Human Resources are an essential part of a SOAR platform. Security analysts, security engineers, forensic experts, network administrators, security administrators, directors of security operation centers, including security orchestration designers and security orchestration and automation engineers, are considered human resources for a SOAR platform [62, 84, 100, 104, 107, 112-114, 116, 133]. Demisto considers any security team who perform day to day security operations as a human resource [147]. An organization must have experts to assess organizations' security infrastructures. According to a report by NSSLab, [62], the security architecture can ensure the organization's security for an assigned level across the entire threat defense life cycle by assessing organizations' existing security infrastructures. Before setting up an orchestration process, the security orchestration designers need to communicate and work closely with security analysts to make sure that the orchestrated process is well-understood [65, 125, 129, 131]. Security teams are the ones who perform coordination, timing, moderation, prioritization and enforcement algorithms for policies based on organizational requirements [62]. Human resources must be able to fully leverage the power of a SOAR platform [125].

A human-centric SOAR platform is necessary where the dashboard and planning tools will be used to make automation work. A SOAR platform is built to work as an intelligence assistant of a security team. According to Bruce Schneier, automation is only

possible in an environment of strong certainty, where everything is related to the planning of certain actions and synchronization of activities [56]. On the other hand, an uncertain world needs direct execution, initiative and prioritization commands. He emphasizes that it is not possible to replace humans; rather, humans are required in security orchestration to make the machine intelligence effective for security response actions. Zonouz et al. [82] mention an online evaluation framework to help administrators. For a coordination model, the work reported in Reference [116] proposes four groups of users, where they consider the security and business continuity teams as a group and the administrator as another group. The further two groups are a group of local users and a group of external or corporate users [116]. Security teams vary by size, vertical and expertise, and their perceptions of what an organization needs from threat intelligence [122]. A SOAR platform should be designed in a way that can work with all sizes, maturity levels and groups of security teams.

2.7 DISCUSSION

This chapter has introduced and analyzed the relevant aspects that motivate the need for a SOAR platform. Throughout this chapter, we have identified and categorized existing SOAR platforms. There is an increasing realization that SOAR platforms can enable significant progress towards achieving the goal of security as a service/ utility. Over the years, several technologies, such as SIEM and Distributed Intrusion Detection Systems (DIDS), have been proposed as solutions to the challenge of providing security as a service. However, security orchestration is still in its early stages of development and has significant potential for research and innovation.

One of the areas of research is standardization, as most of the security vendors are coming up with their own SOAR platforms that have proprietary interfaces or plugins to integrate and access different security tools and services. This heterogeneity works as one of the major barriers to large-scale implementation and realization of security orchestration. Hence, SOAR platforms require new levels of collaboration and performance; the solutions also need to be adaptive to organizational structures that are quite dynamic these days. According to a report by Research and Markets [156], by 2021, the security orchestration market price will hit 1.6 billion USD. A SOAR platform needs to engage security teams fast enough to make a significant difference in the response

time. Security orchestration needs significant amount of research to create results for immediate incident response applications to unforeseen cybersecurity events. This review has enabled us to assert that large scale empirical studies of SOAR platforms and practices under real circumstances will greatly benefit those efforts aimed at addressing the obstacles to security orchestration in different organizational settings.

2.7.1 Open Issues in Security Orchestration

This chapter constitutes a first step towards reaching a common consensus as we examined several state-of-the-art and state-of-the-practice SOAR platforms and compared them with the existing literature. Our review has revealed that the existing SOAR platforms suffer from several open issues. We have analyzed the open issues from three key aspects of security orchestration: people, processes and technology, as shown in Figure 2.10.

- Security orchestration is mainly aimed at increasing automation of security activities that primarily rely on human expertise. The humans need to be involved in the loop of orchestration and automation. With automation, security orchestration requires experts who can easily take the benefit of the automated decisions and take control when automation is inappropriate. There needs to be significant collaboration among different level of staffs involved in dealing with the security orchestration processes and technologies, as each team may have different responsibilities, priorities and metrics.
- Whilst security orchestration and automation efforts are based on scenarios known to security practitioners, security vendors and organizations need to develop and deploy more formal workflows and playbooks for a SOAR platform. Experts involved in the process of orchestration and automation require proper training to gain a common understanding of the workflow, tools and techniques. An organization requires a security architect who can ensure the involvement of risk management and guidance to managed policies. Though one of the motivations behind security orchestration is to handle the collaboration among stakeholders and security experts, a SOAR platform itself requires strong collaboration among business risk owners, risk assessment teams, security operation centers and IT infrastructure managers. The analysis of our review has identified that the current

security industry lacks training related to secure practices. That means that organizations and the security community both need to train current and future staff to keep pace with the wide adoption of security orchestration platforms and conceptualize the data needed to acquire the required insight into security events.

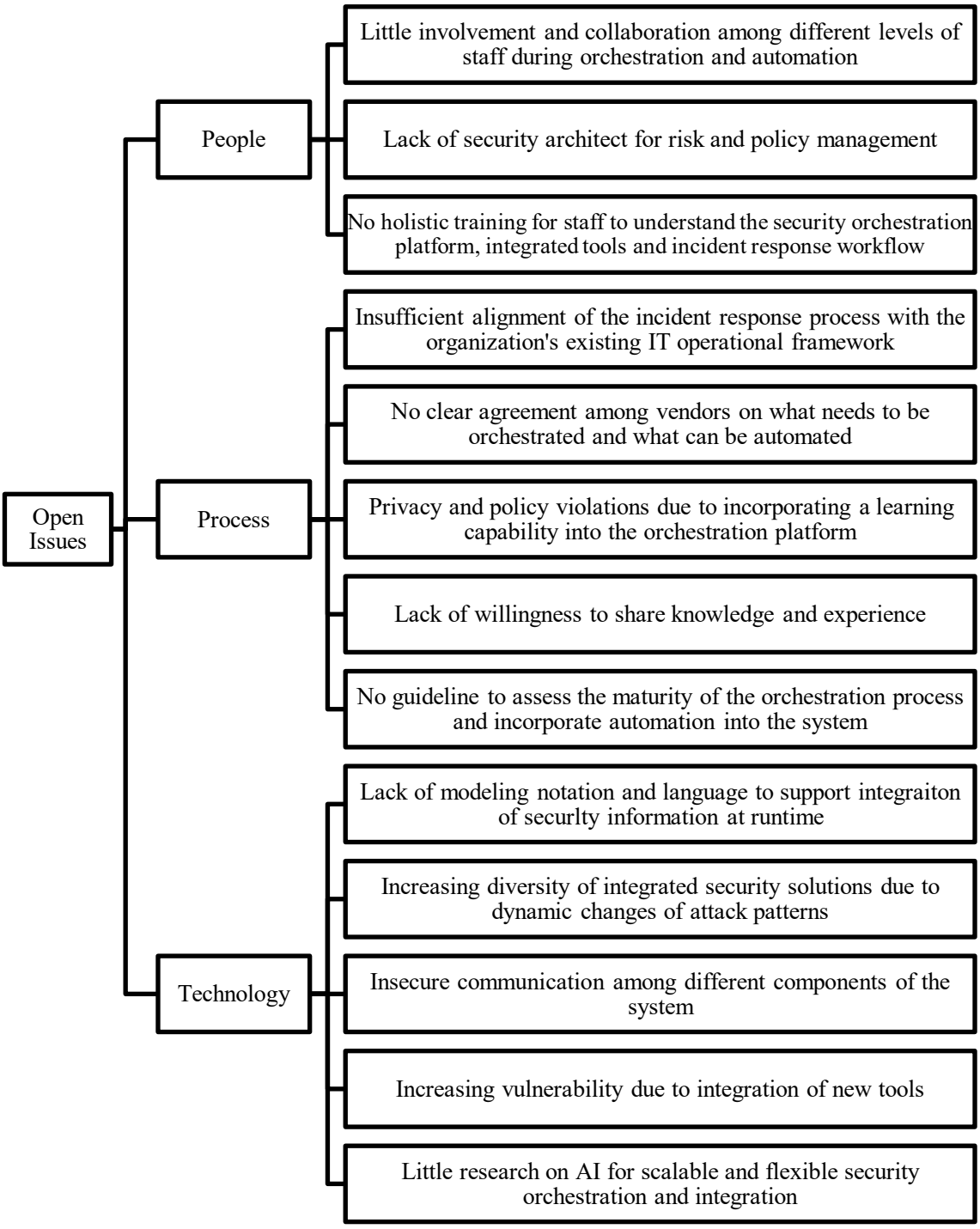


Figure 2.10 Open issues in security orchestration platforms

- The incident response process must be aligned with an organization's existing IT operation framework. An organization needs to have a clear idea of what they can automate and what they need to orchestrate. Hence, there is no agreement on what to automate and what to orchestrate. Nevertheless, some research also refers to orchestration but does not specify its meaning: the focus is often on building plugins to integrate with existing tools. A SOAR platform needs to access organizations' policies and other security tools data to make relevant decisions. Whilst the security team is empowered to streamline incidents, including addressing the issues raised by a SOAR platform, most organizations do not share their threat intelligence with others. This situation can lead to trust issues among organizations.
- Whilst there is an increasing recognition of the importance of security orchestration and automation, the practice of security orchestration and automation is unbalanced. Technology should reach a level such that is able to further support the development of agreement on the definition of orchestration and automation in cybersecurity space. With the advancement of technologies, new vulnerabilities are found and exploited every day. The dynamic change of attack patterns causes the increase in diverse security tools. A SOAR platform should be adaptable with emerging technologies. As stated in the previous sections, orchestration in cybersecurity constitutes an interdisciplinary research area that adopts concepts from research in cybersecurity solutions, SIEM, cooperative IDS, distributed IDS and orchestrated and automated incident responses. There is a need for significant research into modeling notations and languages to support the integration of security-relevant information into streamlining incident response workflows at runtime. There has been no systematic approach to provide a standard API to perform such integration and handle communication among different components of a SOAR platform. There are very few SOAR platforms that provide plugins or support for all the existing multi-vendor security tools.
- One key challenge is to secure the integration and communication of security tools. Another area of future research and development is the application of AI into security automation technology, which can extend, and/or replace where possible, human cognitive processes for security decisions. The existing SOAR platforms are

not currently scalable and flexible enough to handle the heterogeneity of security team structures, sizes and expertise levels.

2.7.2 Architecture Level Support for Security Orchestration

Security orchestration is an emerging area of research and practice. There is little accumulated knowledge and experience available to support industrial decision-making for different aspects of security processes and tools for security orchestration. Siemplify [151] suggested the importance of having a delicate balance between human intervention and automation. New security tools are expected to be adaptable to the existing SOAR platforms. However, a centralized SOAR platform usually incurs huge overhead costs and can be a single point of failure. Due to the ubiquitous realization of cloud, edge, fog and mobile computing, we need to make suitable changes to the ways of deploying SOAR platforms, considering certain properties such as context specifics, knowledge sharing, self-reinforcing and dynamicity. The services provided by SOAR platforms should be fragmented into siloes. Each silo should perform multiple actions in parallel to unleash the best results and act against a threat without delay. This requires choosing an appropriate architecture; for example, microservice, layered, service-oriented, monolithic and so forth.

A SOAR platform needs a well-designed and rigorously evaluated architecture that can support easy integration and smooth interoperability of components and tools developed for various domains by different vendors. There needs to be architecture level support for visibility and comprehensibility of the functionalities and interactions of different components of a SOAR platform that should operate transparently. A SOAR platform's architecture is expected to be dynamically adaptable to the changing threat environment. The MLR has identified the essential components of a SOAR platform that is expected to have certain quality attributes, such as usability, interoperability and flexibility. We can conclude that there is an urgent need to conduct research into identifying and leveraging suitable architectural styles/patterns when designing and evaluating architectures for a SOAR platform.

2.7.3 Limitations of This Review

The MLR we have conducted has some potential limitations. Since security orchestration is an emerging paradigm with mixed and inconsistent terms, the search string used to identify the relevant papers may not have included some words that might be used for security orchestration. The inclusion and exclusion criteria used to assess and select the reviewed studies have been defined by the research team. The focus of this review does not include an in-depth discussion of the limitations of the reported solutions.

The findings of this review are based on the studies related to security orchestration that were found during the time period of January 2007 to July 2017. New technology and platform may appear after that time period. Nevertheless, to determine the research trends on the topic of security orchestration from 2017 to October 2020, we ran our search string on the considered three databases (i.e., IEEE Xplore, ACM DL and Scopus). The number of articles that were found from these databases for the two time periods are shown on Table 2.7. Table 2.7 also reports the number of results that were returned by Google search engine for search string "security AND orchestration" on November 2017 and October 2020. Table 2.7 shows that increasing number of articles have been published in 2017 to July 2020. It also shows substantial increase in the number of grey literatures published between 2017 to 2020. Growing focus on security orchestration from both academic and industry emphasizes our findings that incorporation of SOAR technologies and platforms in a SOC environment is on the rise.

Table 2.7 Number of papers that were returned during 2007 – July 2017 and 2017 – October 2020 for our proposed search string

Source	**2007 – July 2017**		**2017 – October 2020**		**Total**
	# of articles	%	# of articles	%	
IEEE Xplore	600	77.22%	177	22.78%	777
Scopus	1017	57.92%	739	42.08%	1756
ACM DL	271	65.30%	117	28.19%	415
Grey literature					
Source	**November 2017**		**October 2020**		-
	Results		Results		
Google Search Engine	45900		249,000		

We encourage the reader to take the above-mentioned limitations into consideration while using the findings from this chapter. Moreover, some organizations'

security orchestration requirements may not fully be met by any of the reported security orchestration technologies.

2.8 CHAPTER SUMMARY

Security teams may become overwhelmed by the task of monitoring and handling an increasingly huge pool of security alerts generated by a diverse set of security tools. Hence, they may fail to act in a timely manner to deal with security incidents due to the manual and repetitive job of receiving and combining security alert information from multi-vendor security tools. A SOAR platform aims to support security teams to monitor monitoring and deal with security incidents effectively and efficiently by enabling coordination and collaboration among the heterogeneous independent security tools. Integrating and orchestrating the various activities of security tools in an organization needs a comprehensive view of a SOAR platform. Recently, all sort of organizations have started taking interest in adopting SOAR platforms. However, academic research is yet to catch up with the increasing trend of technological innovation and practical adoption of SOAR platforms. Security tool vendors do not share a common/similar understanding while developing and supporting tools, process and technologies for SOAR platforms.

We have systematically selected and rigorously analyzed the SOAR platforms provided by various practitioners and researchers to gain a good understanding of this emerging paradigm. We have also explored the challenges and possible future trends for security orchestration research and practice. Our review has addressed three research questions: (i) What is Security Orchestration? (ii) What challenges is security orchestration intended to solve? and (iii) What types of solutions have been proposed? We have identified and analyzed critical aspects of SOAR platforms found in 95 papers, which were selected based on a pre-designed review protocol. To the best of our knowledge, this MLR can be considered as the first attempt towards systematically reviewing and analyzing the literature on security orchestration.

The analysis of the extracted data to answer RQ1 (i.e., What is Security Orchestration?) enabled us to explore several definitions of security orchestration provided by practitioners and come up with a working-definition for the research on the

topic of security orchestration. The definition of security orchestration provided in this chapter is expected to help practitioners and researchers interested in this topic. Most of the reviewed literature has considered security orchestration as a platform that integrates and unifies various security tools and activities for prompt response to security incidents. The review has identified the key functional and non-functional requirements of a security orchestration platform. Our analysis of the identified functional requirements has revealed three key areas of focus for security orchestration: (i) unification, which is to unify security tools' activities (ii) orchestration, which relates to the process of translating complex processes into streamlined workflow and (iii) automation, which is the process of selecting suitable courses of action to enable automated incident responses. Our review has also identified the key components of a SOAR platform.

A SOAR platform is expected to address several technical and socio-technical challenges for which the review has identified the key techniques, tools and strategies. We have proposed a taxonomy for a security orchestration platform from five key dimensions: (i) execution environment, (ii) automation strategies, (iii) deployment type, (iv) task mode and (v) resource type; which is further split into sub-dimensions. This taxonomy gives a perception of the multidisciplinary nature of a SOAR platform. An organization can compare several security orchestration solutions using the reported taxonomy, which can provide security practitioners with insights into SOAR platforms' usability in different but interdependent processes.

Research is required into designing suitable architectures for supporting the activities of humancentric SOAR platforms, whose common and variable layers can be known to security tools developers and integrators, who are responsible for integrating a diverse set of security tools into a SOAR platform. Such an architecture will also help security experts to decide where to automate incident response processes and where a security orchestration engine is required.

Chapter 3

Security Orchestration and Automation Architecture

Chapter 2 shows most SOCs leverage a number of security tools to detect, thwart and deal with security attacks. One of the key issues of SOC is to quickly integrate security tools and operational activities. To address these challenges, an increasing number of organizations are using SOAR platforms, which are mostly designed in an ad-hoc manner. In chapter 2, we observed that the existing SOAR platform design lacks suitable architecture support. This chapter presents our work on architecture-centric support for designing a SOAR platform. Our approach consists of a conceptual map of SOAR platforms and the key dimensions of an architecture design space. This chapter demonstrates the use of the approach in designing and implementing a Proof of Concept (PoC) SOAR platform. We also report a preliminary evaluation of the proposed architecture support for improving a SOAR's design.

3.1 INTRODUCTION

The adoption of Security Orchestration, Automation and Response (SOAR) platforms has recently gained major popularity among security analysts, Security Operation Centers (SOC) and incident response teams [21-23, 59]. SOAR platforms enable integration, orchestration and automation of the activities (e.g., blocking IPs, scanning endpoints and isolating hosts) performed by security tools and human experts [22].

Chapter 2 shows that existing SOAR platforms lack proper abstractions for designing a platform at the architectural level. Most of the existing SOAR platforms are implemented in an ad-hoc manner, without much attention to the underlying

infrastructure [21-23, 34, 136, 143, 157]. As a result, there can be several engineering challenges involved in embedding agility in a SOAR platform [26, 27, 43, 158, 159]. For example, managing interoperability among isolated and heterogenous security tools with changing environments, integrating new security tools and defining playbooks to adapt with the dynamic changes in attack patterns and advanced technologies and so forth. These challenges result in a highly complex and monolithic design that is hard to evolve. A SOAR's design complexity may also be worsened by a lack of conceptual and practical guidelines for optimal architectural design decisions [22, 158]. The existing SOAR platforms lack any systematic approach to provide a standard set of Application Programming Interfaces (APIs) to integrate security tools or the activities performed by individual security tools or enabling interoperability among different security tools.

An architecture-centric approach [44, 45, 160] is expected to help in reducing the design complexity of a SOAR by modularizing the functionalities and non-functional requirements. The architectural design decision provides a foundation for analyzing and understanding the sub-optimal design choices [44], which can be improved by leveraging suitable architectural styles and patterns. Design space is required to capture and characterize design decisions for integrating techniques and tools that underpin a SOAR platform [22]. Developing design spaces for different domains of software systems is a growing trend [44]. The design space of a SOAR platform involves many architectural design decisions and trade-offs that are impacted by the security tools and applications integrated into these platforms. We propose a concept map considering the functionalities performed by a SOAR platform. It enables the modularization of the functions and separation of the concerns of the components that provide the design space of a SOAR platform.

In this chapter, we present an architecture-centric approach to design and implement a SOAR platform. The proposed approach consists of three parts:

- **Abstraction to model a SOAR platform design space**: We provide a concept map of a SOAR platform that defines and relates the key concepts of SOAR to support understanding of security tools' integration and orchestration. The design space is useful for understanding and analyzing the requirements of emerging SOAR platforms and integration technologies for faster responses and efficiency.

- **Layered Architecture for SOAR platforms**: We provide a layered architecture that modularizes the components into different layers based on two key functionalities: integration and orchestration. These two key requirements are to guide architects to design and deploy a SOAR platform to integrate security tools and orchestrate activities based on integrated security tools. We further consider the architecture style and pattern as a means for delimiting the design space.
- **Proof of concept SOAR platform support**: We have developed a Proof of Concept (PoC) SOAR platform that has been designed to fulfill the quality requirements: *integrability*, *interoperability*, *interpretability*, *usability* and *modifiability,* following the proposed architecture. We have used seven security tools with different capabilities. The evaluation results show the feasibility of the proposed architecture approach for (i) automated integration of security tools and (ii) automated interpretation of incident response activities.

This chapter is organized as follows. Section 3.2 introduces a concept map of a SOAR platforms' design space. Section 3.3 presents the modularized architecture of a SOAR platform. Section 3.4 details the dimensions of a SOAR platform's integration design space. Section 3.5 presents a case study. Section 3.6 demonstrates an evaluation of the PoC. Section 3.7 discusses related work and section 3.8 concludes the chapter.

3.2 SECURITY ORCHESTRATION AND AUTOMATION

SOAR platforms are integrated solutions for an organization's SOC. The underlying technologies of SOAR platforms are designed to interweave people, processes and technologies. In a SOAR platform, people are responsible for intelligence-based decision-making and technologies are used to streamline complex processes. The key purpose of a SOAR platform is to power automation through orchestration. The functionalities of a SOAR are mainly categorized into integration, orchestration and automation [22].

The development of any SOAR platform first needs to focus on *integrating* the security tools in a single platform. Depending on the organization, the security tools can be open source, commercial, proprietary, packaged or even legacy scripts. Security tools are generally integrated using plugins, scripts, APIs and modules. Mostly SOAR vendors

provide plugins and API-based support for 150 – 200 security tools [143, 161]. Security tools generate data in a variety of formats. Furthermore, the data are unified to enable *interoperability* among security tools.

The second key task of a SOAR is *orchestration*. It allows organizations to deploy and operationalize their security process or Incident Response Process (IRP) using a piece of code or script, also known as a playbook. An IRP is a set of activities performed by security experts and security tools. Playbooks contain a set of instructions that make security tools interoperate in a manner whereby the output of one tool is used as an input to other tools. An orchestration process improves the response to a security incident by reducing the manual and repetitive tasks done by human experts.

The third task of a SOAR is *automation* or *response*. An organization needs to identify what they need to orchestrate and what can be automated. Mostly validation, prioritization, reducing false alarms and checking for access control authorization are the different types of activities that are automated through orchestration processes. The SOAR community has not quite reached a consensus on any standard mechanism of automation of security activities.

The following subsections (section 3.2.1 and section 3.2.2) present the key functional and non-functional requirements that we have considered for designing and implementing a humancentric SOAR platform.

3.2.1 Functional Requirements of Security Orchestration and Automation

We consider two core functional requirements of a SOAR platform for integrating security tools and streamlining the incident response process. We adopt the functionality of a SOAR platform outlined in section 2.3.2 of chapter 2.

3.2.1.1 *SOAR as a unifier or hub*

We consider a SOAR platform as a hub that unifies the activities of security tools and provides a single pane for supporting the operations of a SOC. Security tool integration is one of the most important resource-intensive and time-consuming activities in a SOC. Security tools can be integrated using several architectural integration styles [162]. Semantic technology can be leveraged for integrating security tools. A semantic integration mechanism ensures that a SOAR platform can interpret the data consumed

and generated by security tools for interoperability. A SOAR platform first needs to integrate the security tools and then, based on integration mechanisms, it interprets the IRPs. It can enable organizations to use playbooks from different vendors to model an orchestration process by unifying the semantics provided in playbooks. Most SOAR platforms filter incoming alerts based on their syntactic and semantic correctness before delivering them to analytics tools. A SOAR's architecture should support semantics integration among the artifacts produced and consumed by the security tools.

3.2.1.2 *SOAR as a coordinator or orchestrator*

A SOAR platform orchestrates the security tools' activities and streamlines complex security processes into simplified processes. The orchestration processes can be considered as a sequence of actions, where the output of one tool needs to be the input of other tools. A simplified process is easy to follow and enables a SOC to differentiate between manual and automated processes. It also helps to keep track of the ongoing scans and activities that require immediate human involvement. It should be noted that several pieces of literature about SOAR tend to use integration mechanisms or connecting tools as an umbrella term to cover all processes that happen under the banner of security orchestration. Whilst this abstraction is helpful to gain an initial understanding of security orchestration, we argue that architects would benefit from a more modularized model that clearly distinguishes the activities related to integration, orchestration and automation within SOAR platforms.

3.2.2 Quality Attributes Requirements

A SOAR platform should satisfy certain quality attributes requirements or Non-Functional Requirements (NFRs). The essential quality attributes requirements of a SOAR are categorized into design time and runtime requirements. To design the architecture of a SOAR platform, we focus on the following NFRs.

- **Integrability**: Andersson et al. have defined "the ability of a system to easily integrate separate systems or components of a system" as integrability [162]. Security tools integrated into a SOAR platform come from different vendors. An architecture of a SOAR platform is expected to seamlessly integrate security tools and quickly adapt the modification of security tools' functionalities.

- **Interoperability**: According to Bass et al. “interoperability is about the degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context”. A SOAR platform should support semantic integration of different types of artifacts generated by security tools and data sources. The integration mechanism needs to ensure that security tools can interoperate with each other. Security tools integrated into a SOAR platform need to have the ability to both syntactically and semantically interoperate with each other. A SOAR platform should be able to semantically interpret the data and artifacts generated and consumed by security tools.
- **Interpretability**: Interpretability is mainly defined as “the degree to which a human can understand the cause of a decision” [163]. In this thesis, for interpretability, we consider both the ability of a SOAR platform to understand security tools’ artifacts and for a human observer to understand the cause of a decision made by a SOAR platform. A SOAR platform has several components with Artificial Intelligence (AI) capabilities (refer to chapter 2). Thus, the decisions taken by a SOAR platform need to be interpretable to security teams.
- **Usability**: Bass et al. have considered that “usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.” Andersson et al. have defined usability as the “effort to learn, operate, prepare input and interpret output of a program” [162]. A SOAR’s architecture needs to be easily understandable, so that a SOC can easily learn and operate a SOAR platform and interpret the input, output and activities of the components.
- **Modifiability:** According to Bass et al. modifiability is all about changes [45]. In a SOAR platform, changes can happen for incorporation of new features, tools, technologies, standards, platforms and so on. Gorton has proposed modifiability as a “measure of how easy it may be to change an application to cater for new functional and non-functional requirements” [164]. A SOAR platform’s tasks depend on integrated security tools, IRPs and emerging threat behaviors, which change continuously. A SOAR architecture should be flexible enough to provide mapping support for security tools and IRPs to adopt the changes.

Maintainability and flexibility are considered as two aspects of modifiability [153]. Maintainability refers to the degree of effectiveness of modifying a product or system by end-users or maintainers of that a system [35, 153]. It is often considered as the ability of a system to support changes. Flexibility is considered as "the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed".

Due to the limited time and scope of the thesis, we only considered the abovementioned quality attributes for integration of security tools and IRPs in a SOAR platform. These attributes are required for the evolvement of a SOAR platform where security tools can be easily integrated and data generated and ingested by these tools can be seamlessly interpreted. Other quality attributes that are mentioned in Figure 2.6 of chapter 2 are equally important, hence will be considered in future research.

3.2.3 Abstraction for Security Orchestration and Automation

Organizations generically deploy and run a SOAR platform on top of existing security tools, information systems and organizational infrastructures to fulfill their security requirements and business needs. An architect must understand the core concepts of a SOAR platform to design and communicate about the orchestration process and required integration and automation technologies with stakeholders and developers of a SOAR platform. The lack of a comprehensive view might result in concept overlapping and ambiguity. To address this issue, we propose a conceptual map to capture the common terminologies of a SOAR. Figure 3.1 shows the conceptual map of a SOAR platform that provides the key elements and relationships among these elements.

A SOAR platform connects a wide variety of security tools that have different capabilities. By capability, we mean the features and characteristics of security tools, which can support different types of activities. Security tools are generally categorized as detection, analysis and response tools depending on their capabilities (Figure 3.1). This categorization is made based on the activities performed by security tools while responding to an incident. For example, monitoring tools can be considered under detection or analysis tools depending on their contribution to an IRP. A detailed description of the security tools used for this research is beyond the scope of this chapter.

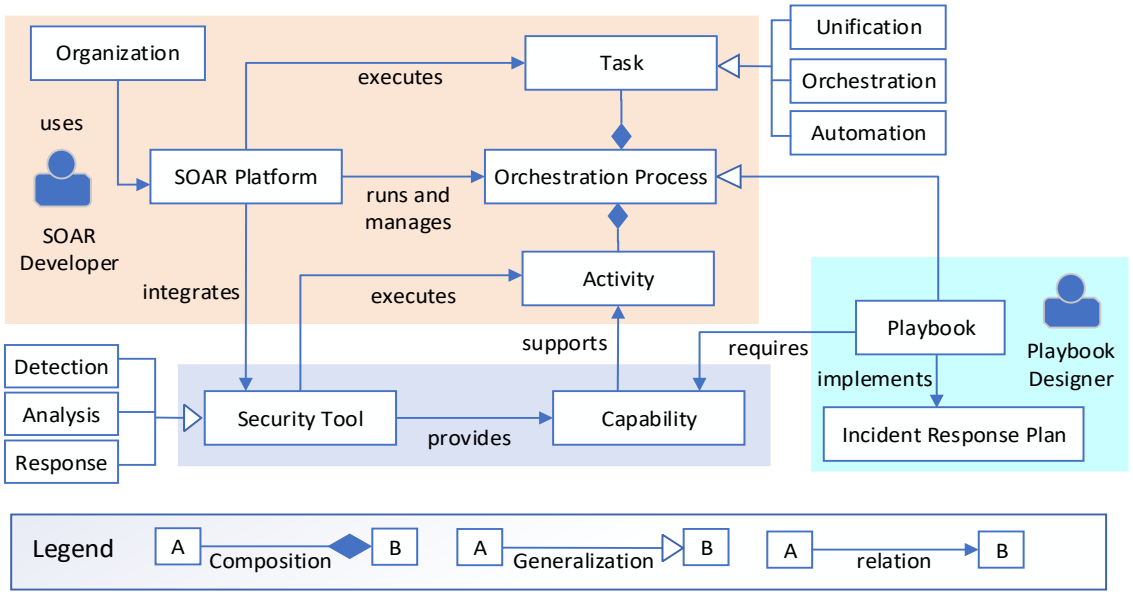


Figure 3.1 Conceptual map of security orchestration and automation

A SOAR platform is designed and deployed based on an organization's security requirements and the available security tools. A SOAR developer needs to design and develop different types of integration mechanisms (e.g., APIs, plugins or modules) to integrate security tools (Figure 3.1). A SOAR platform performs a set of tasks that can be categorized under unification, orchestration and automation. It runs the orchestration process that invokes security tools to perform certain activities. An orchestration process is the composition of tasks performed by a SOAR platform and activities performed by security tools. It contains the invocation actions, scripts to invoke tools and the responses of security tools. Orchestration processes govern the integration, orchestration and automation tasks to respond to a security incident.

The orchestration process is primarily designed in the form of a set of playbooks, which are generally dedicated to a particular security incident and have a dedicated set of security tools that are deployed in an organization's environment. Most organizations also have dedicated Security Incident Response Teams (CSIRT) who mainly design IRPs for security incidents based on an organization's preferred security requirements (i.e., confidentiality, integrity and availability), policies and quality requirements. SOAR developers or playbook designers design and develop playbooks based on the available security tools and well-known integration mechanisms.

3.3 SOAR ARCHITECTURE

We propose an architecture to ensure the functional and non-functional requirements of a SOAR platform. The key research objective is "*how software architecture can play a role in improving the design practices of a SOAR platform?".* We design the architecture of a SOAR platform at two levels of abstraction. The architecture is first designed following the layered architectural style, which provides the first level of abstraction. There are six layers: (i) security tools, (ii) integration, (iii) data processing, (iv) semantics, (v) orchestration and (vi) User Interface (UI) layers, as shown in Figure 3.2. Each layer has both logical and physical aspects. The logical aspects cover the architectural building blocks and design decisions of a SOAR platform. The physical aspects include the realization of the logical aspects by using organizations' technologies and products. Each layer has a separation of concerns that allows security teams to freely choose their preferred components and deploy a SOAR based on their requirements.

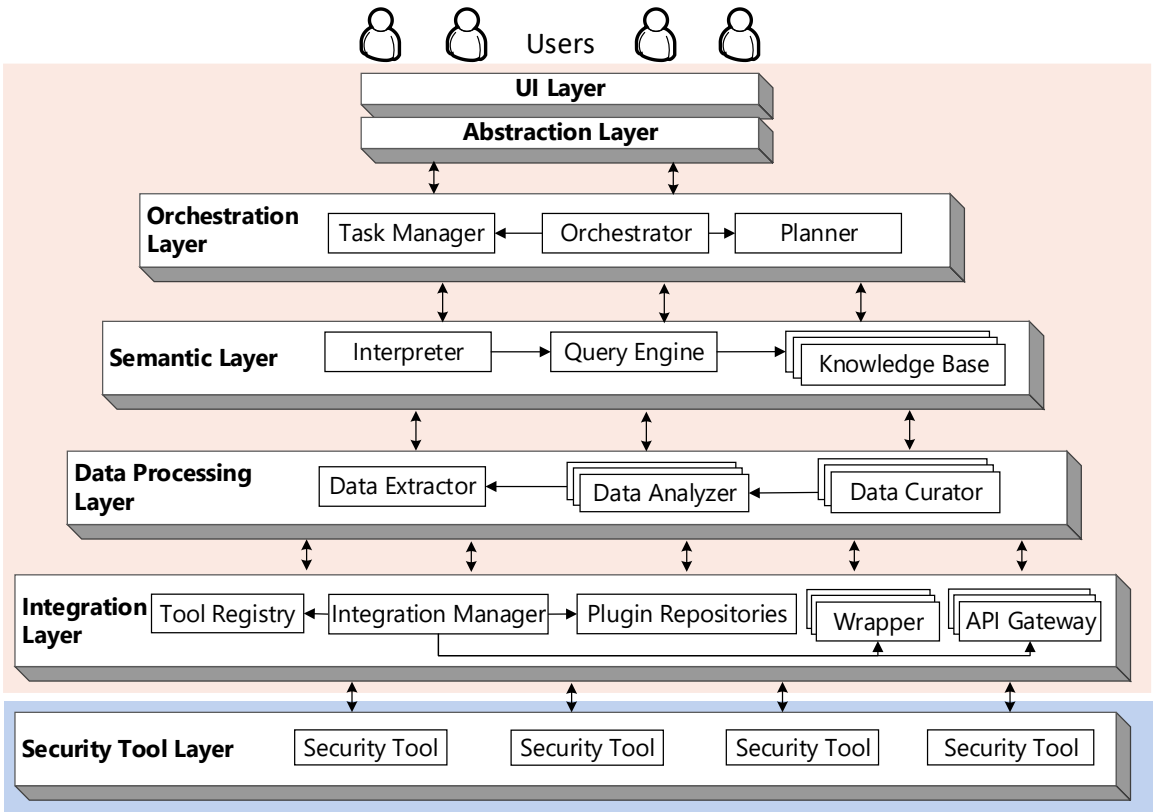


Figure 3.2 High-level architecture for a SOAR platform

Each layer is decomposed into components and sub-components. We consider the components as the lower level of abstractions. Figure 3.2 shows the core components and

interactions among the components that are required to achieve the desired goals of a SOAR platform. Different functionalities of a SOAR platform require different combinations of these components. We specify the components as a principle computation element that implement different tasks of a SOAR to execute IRPs.

3.3.1 UI layer

Security teams initiate existing IRPs or define new plans using a SOAR's User Interfaces (UIs) such as interactive dashboards or Integrated Development Environments (IDE), or Command Line Interfaces (CLI). The UI layer supports *flexibility* in designing UIs that help define IRPs and integrate security tools. A SOC can easily learn and operate a SOAR platform using the UI. An abstraction layer or API layer is implemented as part of the UI layers to maintain and encapsulate the interaction among a SOAR's users and its components (Figure 3.2). The abstraction layer hides the inherent complexity of SOAR platforms from security teams. To increase the *usability* of a SOAR platform, the APIs of abstraction layers need to be easily interpretable and understandable by security teams, so that they can modify, update and interact with a SOAR to provide commands and execute IRPs.

3.3.2 Orchestration layer

The *orchestrator* and *task manager* together form the coordinator of a SOAR platform (Figure 3.2). The orchestrator is responsible for coordinating and forming configurations to achieve *interoperability* and automate the execution of IRPs. The *planner* in the orchestration layer has a set of '*playbooks'* to automate the execution of an IRP and keep track of the tasks being executed. Each playbook has a set of tasks that contain the details of the process about the input required to execute a task and the output that is generated after task execution. The playbooks further contain the conditions that trigger the execution of a task. A playbook's tasks vary depending on the requirements of a SOC and the types of security tools available. The *orchestrator* monitors the successful or unsuccessful execution of tasks. The planner provides a set of APIs through which a user can update or modify the orchestration process. An orchestrator may use a set of APIs to govern the execution of an IRP and enable interoperability among security tools.

3.3.3 Semantic layer

The semantic layer is responsible for the semantic *interpretation* of data that flows across a SOAR platform. It consists of a *knowledge base*, *query engine* and *interpreter*. The knowledge base consists of an *ontology* of security tools, their capabilities and the activities of an IRP, which enables the *interpreter* to semantically interpret security tools' capabilities and IRPs' activities. Ontology is commonly used for formalizing semantic knowledge and defining semantic relationships among data. The *query engine* is responsible for extracting data from a knowledge base. In our proposed architecture, we consider the semantic layer to be separate from other layers to give the SOC the *flexibility* to define or modify an ontology without affecting the other components.

3.3.4 Data processing layer

The information used by a SOAR ranges from business-critical data to usage systems logs, alerts logs and malicious activities that are processed by the data processing layer. The *data curator*, *data extractor* and *data analyzer* are the three main components of the data processing layer. The *data curator* gathers the data produced by tools for analysis. This layer contributes toward *interoperability* and *interpretability* by processing the heterogeneous structured and unstructured data of different security tools and playbooks. It is responsible for sharing semantically structured data among different components of a SOAR through an IRP execution process. An architect can incorporate any automation algorithm or data analysis techniques as part of the data analyzer without affecting the other components of a SOAR.

3.3.5 Integration layer

The integration layer has five components: the *integration manager*, *wrapper*, *tool registry*, *plugin repository* and *API gateway*. This layer is designed for seamless integration of security tools to achieve *integrability*. The *integration manager* works as a description module through which security tools are integrated and information is provided to enable *interoperability* among them. A *tool registry* is responsible for discovering and registering available security tools to monitor their status and report any changes. Security tools are registered in terms of their capabilities (i.e., input, output and functions) and types. The *wrapper, API gateway* and *plugins* are intermediary components that provide interfaces to encapsulate security tools for data translation or

imposing orchestration. An integration manager uses these components to initiate a request and become the ultimate recipient of the orchestrator's commands. The difference between the wrapper, plugins and API gateway lies in the security tools' integration and communication protocols.

3.3.6 Security tool layer

The security tools layer consists of multivendor heterogeneous security tools, which are typically a mix of open source, proprietary, custom and commercial-off-the shelf (COT) products. These tools are mainly characterized as unmodifiable components of a SOAR platform. Given most of the security tools are required to interact with each other, an in-depth understanding of the security tools' data structures and capabilities is necessary to integrate them into a SOAR platform and streamline the IRPs.

Figure 3.3 shows an example UML sequence diagram for responding to a security incident that comprises of components from each layer.

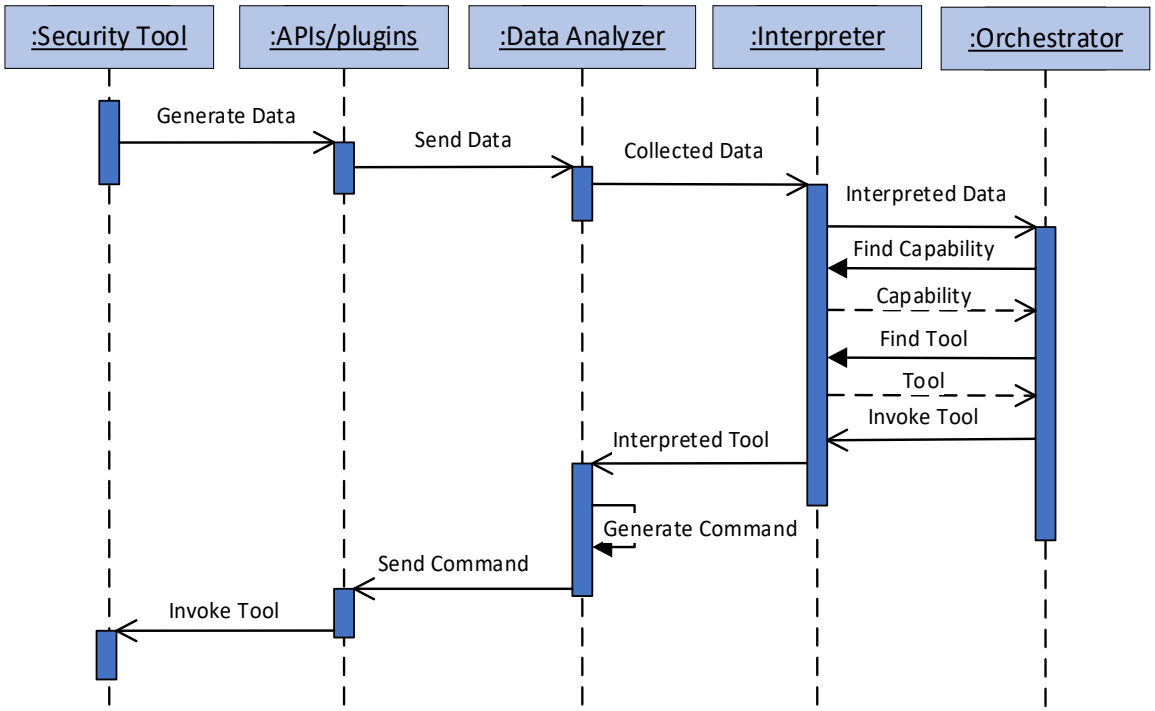


Figure 3.3 An example sequence diagram showing the flow of data and interaction of components

3.4 DIMENSIONS OF THE DESIGN SPACE OF SOAR PLATFORM

The design space of a SOAR platform reveals that the integrated security tools and orchestration process mainly govern the tasks of a SOAR platform. Hence, we have considered the architectural design decisions from the process and technology perspective for automatically integrating security tools and orchestrating IRPs.

3.4.1 Process decision

Along with defining the orchestration process, it is important to define the process for integrating security tools and analyzing data. A SOAR's process varies depending on the mode of a task – automated, semi-automated or manual. The automation of the integration process relies on five design decisions for the *integration process*, *interpretation process*, *security tools to capability mapping process*, *security tool discovery process* and *security tool invocation process*.

A decomposition of the functions based on layers helps in selecting a suitable technology, depending on the required process. For example, the task to manually integrate security tools is separated from automatically interpreting the security tools' data. Security tools are first required to integrate into a SOAR platform, then processes are designed to interpret the security tools' data and the IRP's activities. Here, the modular architecture helps to define different processes, which are mainly the orchestration of the security tools, the SOAR's components and organizational information systems. A SOAR platform can be centralized, distributed or hybrid, depending on an organization's infrastructure [22]. For centralized or distributed applications, the communication protocols are different. In most cases, these communication protocols (i.e., REST API, RPC and event-driven) are hidden under the internal structures of security tools, which expose their functions through APIs. A communication process can be designed to manage distributed communication among different security tools.

3.4.2 Technology decisions

From a technology perspective, we mainly consider *the integration technologies*, *interpretation mechanisms* and *tools discovery mechanisms* that are required for integrating security tools, designing the orchestration process and powering automation.

Chapter 2 shows six automation strategies (section 2.6.2) that are adopted by the existing SOAR platform. An underlying technology infrastructure consists of the assets of an organization, depending on the type of the automation strategy. An example of assets includes the various hardware and software infrastructures (i.e., computer systems, operating systems and applications) that an organization needs to protect from security attacks. Orchestrations can take place in different types of environments, which can be open or restricted. We need to consider different architectural integration styles to ensure that the integration constraints related to different security tools and stakeholders (e.g., semantic, performance and component constraints) are addressed [162].

In the following section, we provide a set of design decisions that need to be made by an architect.

- Building a generic block of a SOAR platform. An architect can choose to design a playbook and script for orchestration and automation.
- Disseminating tools that are integrated and participate in orchestration. Architects have to decide on how to map security tools to the IRP and where to deploy them in an organization's environment so that the orchestrator can invoke the tools when required.
- Setting up a mechanism for an orchestrator to discover security tools. An architect has to choose integration styles and define processes for the discovery of the security tools.
- Setting up and starting an orchestration process. An architect has to decide who has the right to modify the process and provide an interface to modify or add new IRPs.
- Designing APIs for hiding the architectural complexity from end-users. An architect has to design APIs through which end-users can interact easily with a SOAR platform.

Table 3.1 shows a summary of the architectural design decisions for achieving the desired functional and non-functional requirements of a SOAR platform. By architectural design decisions, we mean the design decisions that would have a system-wide impact and/or impact on more than one non-functional requirement [45].

Table 3.1 Summary of the architectural design decisions

Design Decisions	Expected Benefits
Ontology for formalizing security tools and activities of IRPs	Make a SOAR architecture flexible enough to integrate different types of security tools with varied data formats
Use of ontology for semantic integration and information discovery	Support tools' specific integration and automated execution of IRPs in a dynamic environment
Layered architectural style	Easy evolution of the SOAR's components and easy modularization of functionalities and components
Abstractions of SOAR's components tasks with a set of APIs	Make a SOAR platform easy to use, manage and learn for end-users
Automated integration and interpretation processes	Enable reuse of existing components with changes in IRPs and security tools
Share ontology template in a centralized repository pattern	Provide access to the ontology to its end users and support flexibility for updates

3.5 CASE STUDY – PROTOTYPE IMPLEMENTATION

In this section, we present a Proof of Concept (PoC) SOAR platform namely STUn – **S**ecurity **T**ool **Un**ifier) [165] that we have designed and implemented based on the proposed architectural approach. The functional requirements of STUn are to *automate the process of integrating security tools*, *automate the selection of security tools to execute an IRP* and *automate the execution of a set of IRPs*. We designed STUn such that it is easily evolvable for future changes and supports agility with the emergence of new technologies, processes and tools. In this implementation, we considered two types of changes that are most common in a SOAR's execution environment: changes in security tools and changes in IRPs. Figure 3.4 presents the implementation architecture of STUn. We analyzed the instructions for integration and orchestration to select the technologies and identify the design decisions. We designed automated integration processes and selected semantic technologies to enable semantic integration and interpretation of security tools' data. We further identify the requirements for and designed a set of declarative APIs to hide the complexity of the architecture from the security team. Security teams can use the declarative APIs to interact with the SOAR platform.

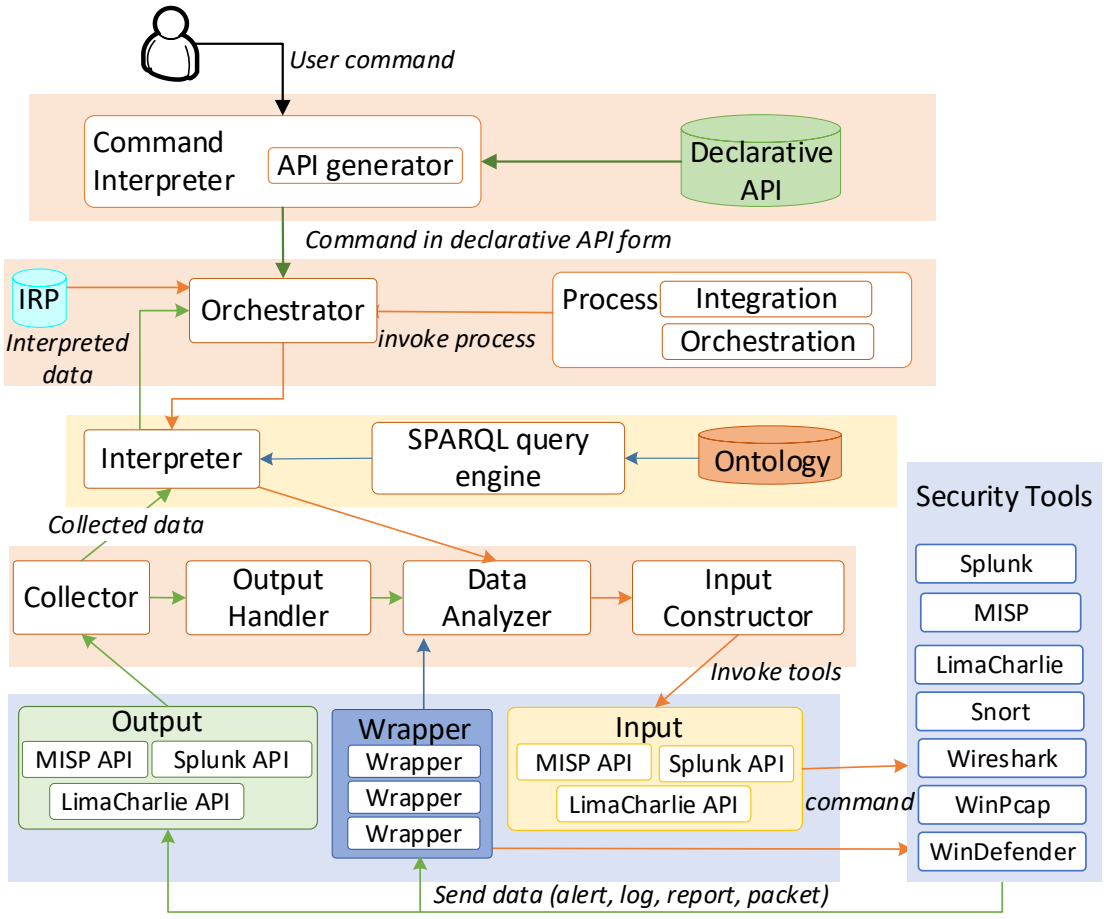


Figure 3.4 Implementation architecture of the PoC for security tool integration

We selected seven open-source tools with varied capabilities. The selected tools are *Snort*, *Splunk*, *LimaCharlie*, *MISP, Windows defender, Wireshark* and *WinPCap* which are IDS (Intrusion Detection System), SIEM (Security Information and Event Management Tools), EDR (Endpoint Detection and Response) tools, Open Source Threat Intelligence and Sharing Platforms (OSINT), Firewall and packet monitoring and logging tools, respectively. The security tools were selected based on the diversity in their capabilities because the execution of an IRP would require multiple security tools.

For instance, Snort is an open-source Intrusion Detection System (IDS). It usually generates alerts upon detection of anomalies or malicious activities in network traffic. It also works as a sniffing tool that can sniff network traffic and perform packet logging. Splunk is a widely-used Security Information and Event Management (SIEM) tool. It performs a variety of operations. Among them, the most popular is to collect log data from various sources, normalize data, correlate the collected data and present the results

to SOC for further analysis. Limacharlie is a cloud-based Endpoint Detection and Response (EDR) tool that operates by running sensors on endpoints. It collects data from different endpoints and enforces detection and response rules. It can send commands to an endpoint to isolate malicious endpoints, terminate or suspend a malicious process, and even delete malicious files from endpoints. MISP is an open-source threat intelligence platform, widely used by security teams for sharing, gathering and storing Indicators of Compromises (IOC) of targeted attacks, threat intelligence and vulnerabilities in a structured manner.

We used 24 different capabilities of the selected tools, with MISP as a new tool to be integrated later. We have curated a set of IRPs from Demisto's (i.e., a SOAR platform provider) collaborative playbooks [166]. We have selected 21 IRPs and slightly modified them to fit the capabilities of the seven security tools used for our research. We designed another 48 IRPs as a new set of IRPs that PoC would be required to execute without user intervention. The list of capabilities and IRPs are available at [165].

The implementation decision incorporated an API-based integration style as our primary mechanism to integrate security tools into a SOAR. The data from security tools such as MISP and Splunk have been made accessible through their APIs. Besides this, we have built wrappers for security tools that do not provide specific APIs such as Snort. Integrating a new tool required us to identify the security tool's APIs, or information sharing protocols, and implement a suitable integration mechanism. The API and wrappers of Figure 3.4 are part of the *integration layer* of the PoC. Figure 3.5 shows examples of the method of the interface of SIEM, EDR and IDS' API. Splunk, Limacharlie and Snort are considered as instances of SIEM and EDR, respectively.

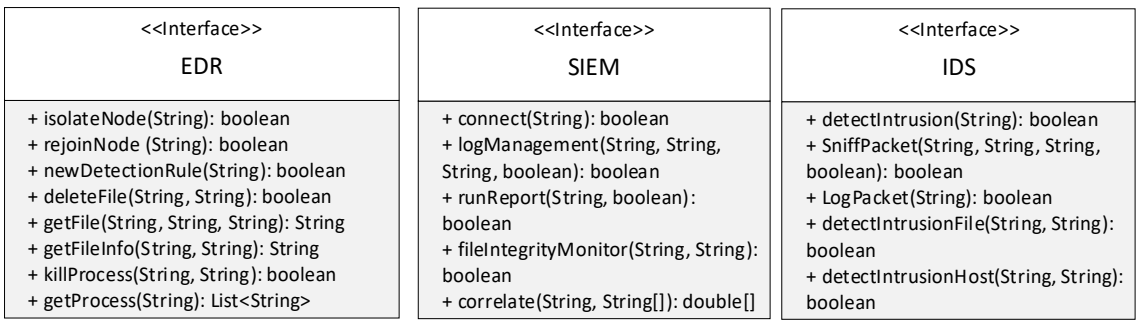


Figure 3.5 Interfaces of EDR, SIEM and IDS in UML class diagram form showing only the methods of each security tool

We also designed an ontology to formalize the security tools, their capabilities and IRP's activities to enable semantic interpretation of the security tools' data. The detailed design of the ontology is presented in chapter 4. Each security tool can execute multiple activities and each activity can be executed by multiple security tools. We used an Apache Jena Fuseki server to store the ontology. Security tools are formalized based on their capabilities and the activities of IRPs are mapped with the security tool class of an ontology. Table 3.2 and Table 3.3 illustrate how the security tools and IRPs have been mapped onto an ontology. We designed a SPARQL query engine to retrieve the required information from the ontology. The retrieve data are interpreted through an interpreter, which mainly deconstructs the data for further processing. The designed ontology, along with the interpreter, built the semantic layer.

Table 3.2. Illustration of a selected set of object properties of the security tool class of an ontology

Security tool	**Security Tool class**	**has Capability**	**Capability class**	**Execute Activity**
snort_s	IDS	intrusion_detection_s	IntrusionDetection	detectIncident
limaCharlie_l	EDR	intrusion_detection_l process_killing_l	IntrusionDetection ProcessKilling	detectIncident killProcess
splunk_s	SIEM	log_collection_s alert_analysis_a	LogCollection AlertAnalysis	collectAlertLog investigateAlert

Table 3.3. Illustration of a selected set of data properties of the security tool class of an ontology

Security tool	**Security Tool class**	**Is Integrated**	**Has InputType**	**Has Rule**	**Has ConfigFile**
snort_s	IDS	True	network traffic	False	snorts.config
limaCharlie_l	EDR	True	Payloads	True	inputs.conf
splunk_s	SIEM	True	Logs	True	LCConf

We built a collector to gather the security tools' data, which are sent to an orchestrator via the interpreter for actions, e.g., Splunk's API is configured to receive system logs of various endpoints. This data is searched and processed to find programs, files or users that could be malicious. To formulate the commands further, an input constructor was built. Figure 3.6 shows the example Splunk and Limacharlie collectors that are instantiated from the collector interface.

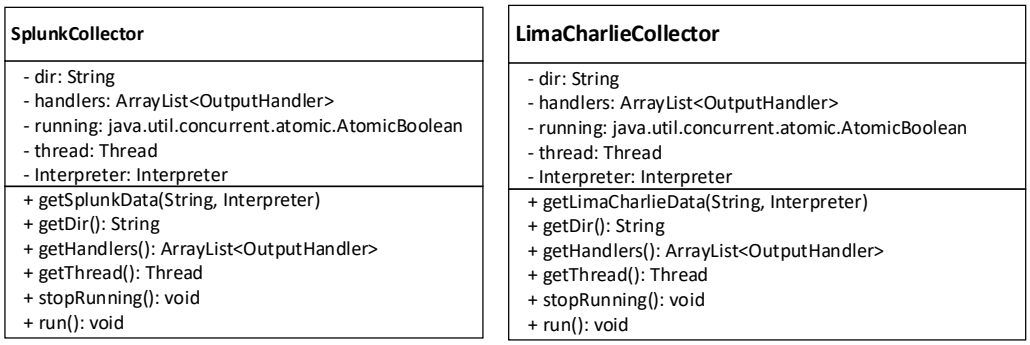


Figure 3.6 Example of Splunk and Limacharlie collector class UML diagram

The automation algorithms or processes have mainly been built as integration processes that are the parts of the orchestration layer (Figure 3.4). We designed and implemented scripts to define the automated integration process, which includes selecting the security tools based on activity descriptions, interpreting their capabilities, formulating the input commands and finally invoking the security tool by calling appropriate APIs. We have presented the detail of the integration process in Chapter 5. We have also shown how the ontology is leveraged to automate the integration process. An example is shown in Figure 3.7, where the output of Splunk is sent to LimaCharlie. The orchestrator is required to collect the output of Splunk and then interpret it. All the data generated by Splunk may not be required by LimaCharlie; so, STUn would be required to construct the input of LimaCharlie from Splunk's output to invoke LimaCharlie.

Figure 3.8 shows an example process where the output of Limacharlie is sent to Splunk. Similar to the above process, the commands are required to construct the input of Splunk from Limacharlie's output to invoke Splunk. We developed and designed this process as part of the integration process to automate the interpretation of the security tools' data, which enables seamless interoperability among the security tools. Using the

integration process, data sharing among the security tools of Figure 3.7 and Figure 3.8 happened seamlessly.

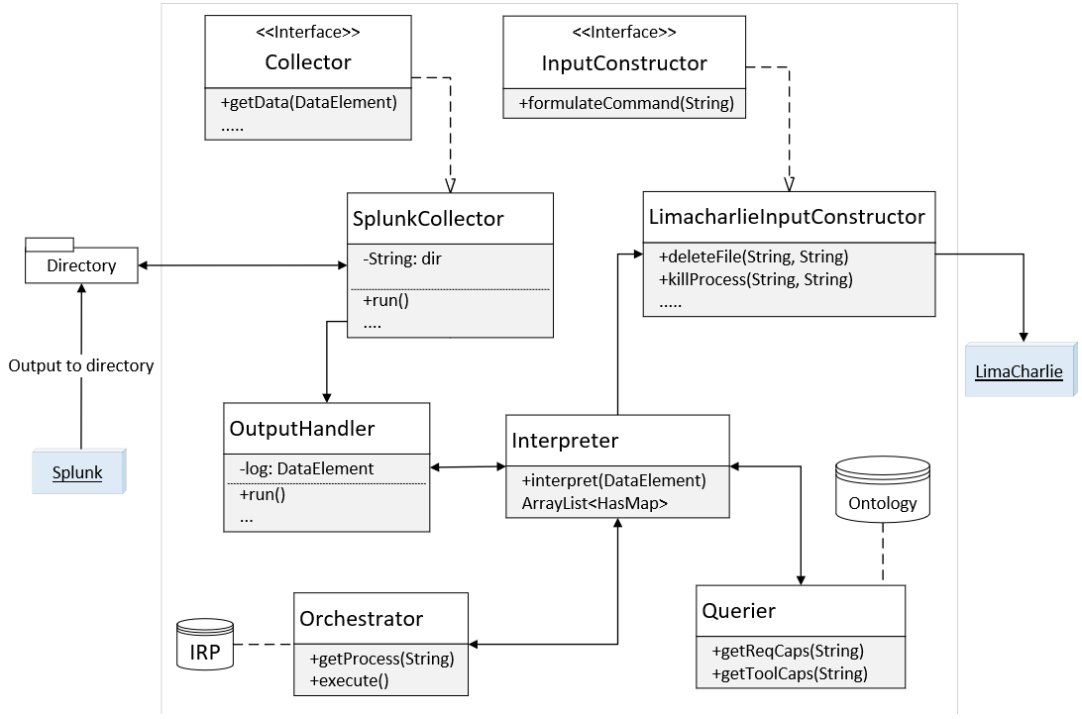


Figure 3.7 Example of data transfer from Splunk to LimaCharlie

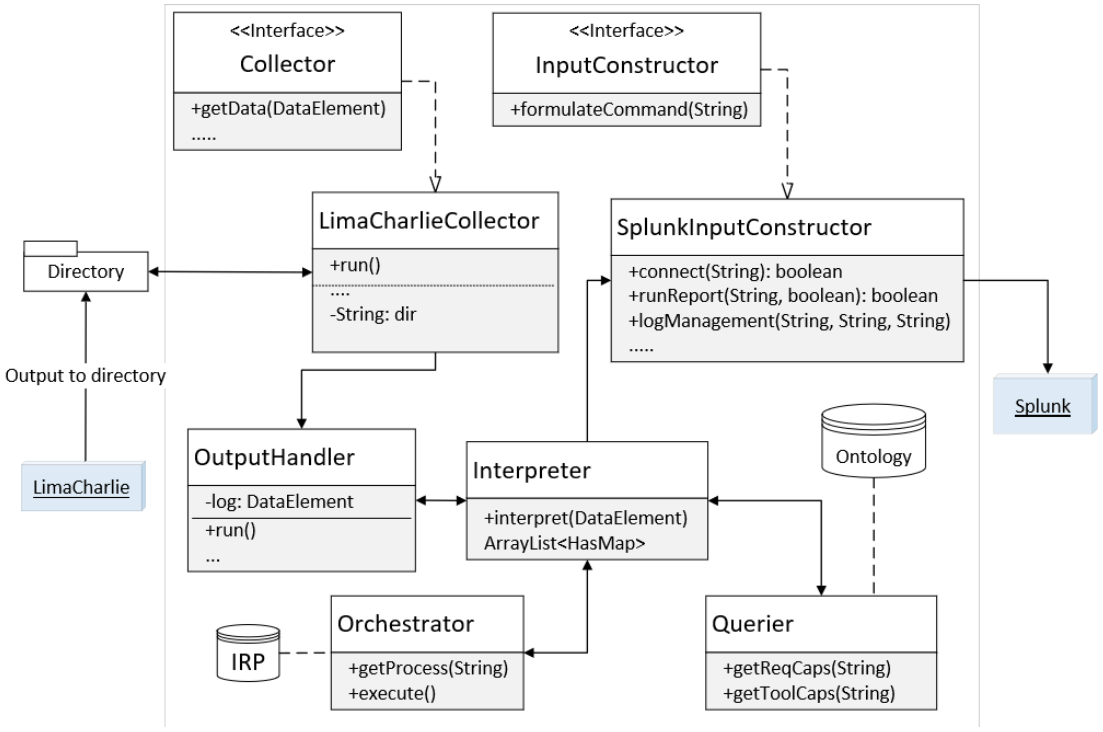


Figure 3.8 Example of data transfer from Limacharlie to Splunk

We present an example to illustrate how STUn enables interoperability among different security tools and automates the response process. We consider an IRP is available to perform “deletion of malicious file” which contains three main steps: collection of system logs, detection of a malicious file and deletion of a malicious file. In STUn, Splunk works as a log analyser that collects a system’s logs and performs correlations among them. It can also monitor a system’s logs and detect malicious files. Specific rules are designed to detect malicious files. Upon detection of malicious files, Splunk generates an alert or report.

Figure 3.9 shows a sequence diagram of how, based on the alerts generated by Splunk, Limacharlie deletes the malicious file, which demonstrates how the components of STUn enable interoperability between Splunk and Limacharlie. As shown in Figure 3.9, the alerts are collected by the collector, which is sent to the interpreter for interpretation of alerts. Features of Splunk alerts or outputs are defined in an ontology. The interpreter invokes the query engine that queries an ontology to interpret the alerts and identify the details of the malicious file. Next, it sends the details to the orchestrator, which finds an IRP is available to respond to a malicious file that is to delete the files. It requests the interpreter to identify the security tools that can delete a file. The details of the security tools are available in an ontology; thus, the interpreter returns Limacharlie to the orchestrator. The orchestrator selects Limacharlie to delete the file and send a message to the input constructor to invoke LimaCharlie. To formulate the commands, the input constructor requires the details of the file that it retrieves from the output handler. Based on the message from the input constructor, the output handler deconstructs the output of Splunk and sends the file details to the input constructor. The input constructor formulates the input commands and invokes Limacharlie to execute the activity, delete file. Upon getting the commands, Limacharlie deletes the malicious files.

We designed a set of declarative APIs to hide the details of the components of STUn from security teams. We proposed a set of declarative APIs through which a user can provide text-based commands or interact with the SOAR through the declarative APIs. The requirements and design of the declarative APIs are proposed in chapter 6. The declarative APIs and commands interpreter were designed leveraging well- known AI tools and techniques, such as semantic technologies and NLP. The declarative APIs

are mainly designed for the security teams to interact with STUn to integrate new tools, update the ontologies, and define or execute IRPs. The declarative APIs and command interpreter of STUn (refer to Figure 3.4) form the abstraction layer of the proposed architecture (refer to Figure 3.2).

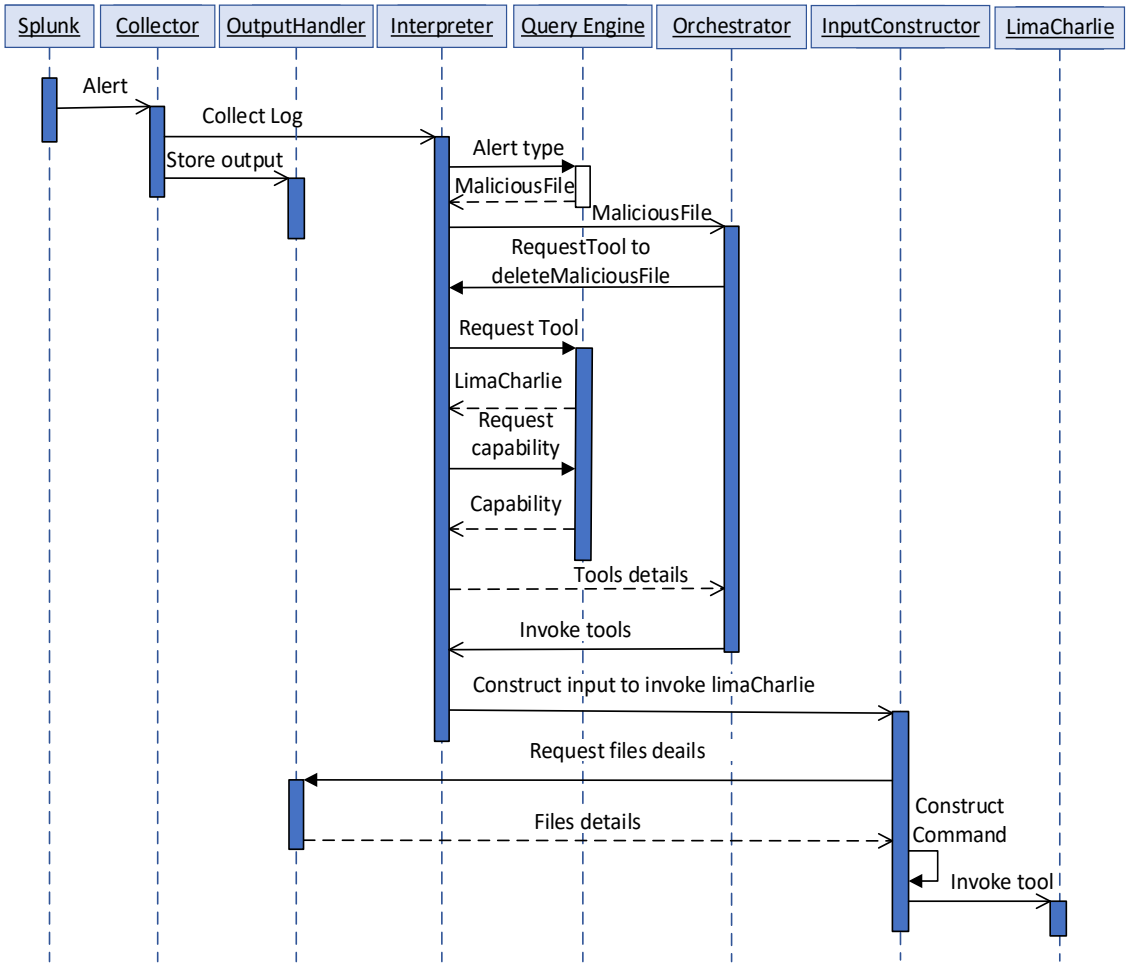


Figure 3.9 Sequence diagram for deletion of a malicious file that is detected by Splunk and deleted by Limacharlie

3.6 EVALUATION

In this section, we report on how the PoC has been evaluated to demonstrate the feasibility of the proposed architecture approach, based on two scenarios.

3.6.1 Automating the Integration of Security Tools

Let's assume that a user has expressed a goal of integrating security tools and we have decided to use the proposed architecture for automating security tools' integration. In the

current implementation, an ontology is available that works as a knowledge base of a set of existing security tools. To integrate the available security tools, the orchestrator provides a template of an ontology to users to specify the tools' capabilities and map them onto the available activities or an activity it can execute. A user can also provide text-based commands or use declarative APIs to integrate security tools. In this chapter, we did not discuss the declarative APIs. The details of the declarative APIs and how the user can provide text-based commands are discussed in chapter 6. This process stores the security tools' information in an ontology that makes the information available to the orchestrator. If the security tools have different capabilities, the information is updated in the ontology. Furthermore, the process for automating the integration of the security tools is invoked, which enables the collector to collect the security tools' output and the orchestrator to formulate and send commands to the security tool for executing the desired activities.

Other integration approaches, such as designing static APIs for communicating with security tools or plugin-based integration, require the development of a wrapper, along with connection with the data curator and the orchestrator, to collect the security tool data. The collector needs to be configured to access the data generated by security tools. Thus, integrating a single tool would require the development of at least one component and connection of that component with the orchestrator. For a security tool with multiple capabilities, for instance, Splunk and Limacharlie have different sets of APIs to invoke different capabilities, a single API or wrapper would fail to invoke different capabilities.

For example, for LimaCharlie with static API based integration, we have designed two sets of scripts to *kill a process* and *isolate a process*. For seven security tools with 24 capabilities, at least 48 connections are required among the orchestrator and security tools when considering API and wrapper-based integration to take the output and provide the commands to execute an activity. Any increase in the connections and components increases the design space of a SOAR. With the inclusion of new security tools, a new connection emerges, and a user would be required to go through the existing APIs, wrapper and connection to integrate a security tool in a playbook to execute an IRP. An update in the existing security tool features, for example, the addition of new capabilities

or a change in the existing API parameters, also requires the design of connections and updating of the playbook where the security tools have been used.

With the semantic-based integration approach, we only need to update the security tools' details in an ontology. The connections between the ontology and other components have already been designed and do not require any changes. Thus, with the PoC, the number of components and connections remains constant with the integration of new security tools: that is MISP. Without considering the proposed architecture approach, the number of components would increase by at least 2 upon integration of new security tools. We found that semantic-based integration is more suitable in this case. This demonstrates that the proposed architecture-based implementation keeps the number of components and connections lower by reusing the existing components.

Our observation from running the experiment reveals that building wrappers and APIs requires more time than updating the security tool details in an ontology. Hence, ontology-based automated integration processes free up the SOC's time.

3.6.2 Automating the Interpretation of the Activities to Execute an IRP

We assume a user has expressed his/her goal is to identify and isolate suspicious endpoints. Using the current implementation, the orchestrator can identify the capabilities required to execute the activities and then select the security tools that can execute that capability. As the process for automatically identifying the capabilities required to execute an activity and selecting the security tools are already defined, a user would not be required to manually identify the security tools. He/she simply needs to request the orchestrator to give them access to those security tools that can perform the required activities. The orchestrator runs the process and returns the available security tools. Then the user can also define which security tools should be used for each activity. Next, the orchestrator automatically generates the commands to invoke the security tools to execute a sequence of activities. In this whole process, the current architectural-based implementation has reused the existing process, components and protocols.

With the non-modular and monolithic implementation of a SOAR platform, a playbook is required to design to fulfill the user's goal. Developing a playbook would require an understanding of a playbook's structure, knowledge of the available security

tools, developing scripts to access the data generated by the security tools and their specific APIs to execute an activity. In the monolithic approach, each playbook is designed for a specific IRP, which cannot be reused even if the new IRP is a subset of the existing IRPs. A user is required to modify the existing playbook to execute the new IRP.

Modularizing a SOAR's architecture provides a clear understanding of which part would require an update and which components can be reused without modification. Reusing the existing components provides the following benefits: a SOC spends less time adapting the changes and the evolution of a system does not increase the complexity of the architecture. Furthermore, it reduces the overhead for users by adopting the changes and providing processes that can be reused. The evaluation shows that, without separating the concerns, the number of changes would require more than our proposed architectural-based implementation.

The PoC has accurately executed 45 IRPs among the new 48 IRPs. For three of the IRPs, the orchestrator could not find any security tools with the required capabilities to execute some of the activities, thus those were executed partially. The successful execution of the 45 IRPs demonstrates that the developed PoC has accurately interpreted the data generated by the security tools being used without user intervention. The security tool MISP is also used by some of the new IRPs; thus, it has also been successfully integrated. From the evaluation, we also observe that incorporating the changes in the PoC is easier than for other approaches.

This chapter has demonstrated the feasibility of the proposed architecture for security tool integration and IRP interpretation based on three quality attributes: *integrability*, *interoperability* and *interpretability*. The details of the realization of the architecture for semantic integration, the automated integration process and encapsulation of architecture complexity through a set of declarative API are presented in chapters 4, 5 and 6 respectively. These chapters also present how the semantics layer has been realized and the challenges associated with building the different components. These chapters also demonstrate how the proposed layered architectural style helps to achieve *usability* and *modifiability* along with *integrability*, *interoperability* and *interpretability*.

Other quality attributes of a SOAR can be evaluated by following different architectural evaluation techniques, such as the Scenario-based Architecture Analysis Method (SAAM) and Architecture Tradeoff Analysis Method (ATAM) [42, 45].

3.7 RELATED WORK

The leading security service providers aim to provide SOAR platforms to deliver end to end security services [32, 143, 161, 167]. For example, FireEye (i.e., a leading cybersecurity company) designs a SOAR platform to integrate its endpoint products and offer support to its industry partners [143]. Meanwhile, the start-ups mainly focus on developing APIs to integrate different third-party solutions and provide playbooks for automated and semi-automated IRPs [35]. The ad-hoc implementations of a SOAR platform increase the design complexity of such a platform as these platforms are built as a whole, without separating the concerns of the deployed components. Furthermore, a SOAR is a large-scale system that integrates an organization's information and security tools. Organizations face several challenges in managing these solutions while any changes occur in the underlying operating environment, such as integrating new security tools and defining new IRPs [22, 159]. Our work addresses these kinds of challenges.

The current state-of-the-practices and state-of-the-arts of SOARs lack a shared understanding between the vendors and stakeholders of SOAR [21, 23, 58, 59, 73]. For example, there is no shared understanding of the key software components and technologies that are necessary to integrate and enable interoperability among various security tools and bring automation to the IRPs' execution. In these studies, the SOAR platform has mainly focused on security tool interactions, isolated processes and low-level infrastructures, while paying less attention to the problems of how different components of a SOAR and security tools coordinate.

A security team requires an understanding of the internal structure of a SOAR (i.e., libraries to integrate new security tools or requirements) to adopt the changes in a SOAR platform execution environment. Adopting the changes remains a tedious and difficult undertaking for end-users. State-of-the-art approaches for security process modeling provide limited or no decomposition mechanisms, which easily results in monolithic processes that address multiple concerns in a single model [21, 23, 59, 73].

None of the existing work provides the architectural design space that could inform architects of the decisions to be made where multiple components are interconnected. Software architecture is composed of early design decisions, which can help to address some of the existing challenges to be addressed by SOAR platform designers [44, 45, 158]. An increased focus on the architectural aspects of SOAR can also facilitate further research on the design decisions of the existing SOAR platforms to form guidelines, rules and design techniques. The rise of security incidents has increased the demand for knowledge, processes and techniques for designing and deploying highly configurable and scalable SOAR platforms. As most organizations prefer to utilize their available software and security tools, it would be helpful to consider architectural design decisions for trade-off analysis before deploying a SOAR platform to enhance a SOC's efficiency.

3.8 CHAPTER SUMMARY

Exploring and understanding architectural design decisions before designing and implementing a SOAR platform is a valuable task. The captured design decisions would help developers as well as the SOC staff of an organization to systemize their decision processes and trade-off analysis. The architectural design decisions would serve as a standalone lexicon to describe and evaluate the existing and new SOAR platforms. In this chapter, we have designed a conceptual diagram of a SOAR platform to support an architect's understanding of the design space of SOAR. We have further identified the requirement of a SOAR in terms of unification, orchestration and automation and proposed a layered architecture to modularize the functions and separate the concerns of the components of a SOAR platform. The architecture design decisions are chosen from process and technology perspectives. We have used the proposed approach to design and implement a PoC SOAR platform for an ad-hoc SOC infrastructure and observe its impact on the automated integration and interpretation process. We have leveraged well-known architectural styles and patterns to implement the PoC. We have observed that consideration of the principal dimension of the architecture design space has improved SOAR design practices.

Chapter 4

Automated Interpretation and Integration of Security Tools

In chapter 2, we observed that a security orchestration and automation platform aims to integrate the activities performed by multi-vendor security tools to streamline the required incident response process. Chapter 3 has proposed a layered architecture for a SOAR platform, considering some of the key functional and nonfunctional requirements of a SOAR platform. To make such a platform useful in practice in a Security Operation Centre (SOC), in this chapter we address three key challenges: integrability, interoperability and interpretability. We proposed a novel semantic integration approach to automatically select and integrate security tools with an essential capability for auto-execution of an incident response process in a security orchestration and automation platform. The capability of security tools and the activities of the incident response process are formalized using ontologies, which have been used for an NLP-based approach to classify the activities for the emerging incident response processes. The developed ontologies and NLP approaches have been used for an interoperability model for selection and integration of security tools at runtime for the successful execution of an incident response process. Experimental results demonstrate the feasibility of the classifier and interoperability model to achieve integrability, interpretability and interoperability of security tools integrated into a SOAR platform.

4.1 INTRODUCTION

Emerging threat behaviors and variations in organizations' infrastructure cause security experts to change the deployment and execution environment of SOAR, such as the

integration of new tools, updates of tools' capabilities or modification of an IRP [21, 23, 74]. Existing SOAR platforms, however, are not adaptive towards such changes [21, 23, 74]. Security teams must sufficiently understand the APIs and rules of SOAR platforms to adapt to the changes by defining new rules or developing new APIs [20, 61, 168]. Human intervention is required to adjust the changes because security tools are not *interoperable* and SOAR cannot *interpret* security tools' activities, their input and generated data [142, 168]. According to a recent report by the Enterprise Strategy Group [169], on average, a SOC has 25 different security tools, and this number goes up to 100 for some SOCs. Most of these tools work independently. The SANS Institute (Escal Institute of Advanced Technologies) has revealed that the integration of security tools is the third most challenging task for SOC [170].

A SOAR platform requires the semantic knowledge to formalize various inputs, outputs and activities of security tools. The formalized concepts enable a SOAR to *interpret* the changes in runtime environment and *automate* the execution of modified or new IRP without any human intervention. Ontologies can be used to provide the required formal specification to support *integrability*, *interoperability* and *semantic integration* of security tools in a SOAR without any human involvement [171, 172]. Semantic integration refers to the ability of a SOAR to understand the semantics of the input or output of security tools. A SOAR can semantically interpret the activities of security tools when the formalization incorporates semantic integration of security tools.

The process of defining a suitable ontology is not straightforward [158]. A well-built ontology depends on domain expertise. Formalizing various security tools and the activities of IRPs is challenging due to the ambiguity of the terminology used by different vendors. The features of security tools and activities are defined using Natural Language; the same activity is defined using different terms in different IRPs. As the development of an ontology is an incremental process, domain experts are required to perform manual tasks to keep the ontologies updated, as per the new knowledge.

We propose an integration framework for SOAR that integrates the data generated by different security tools to automate the execution of an IRP by making security tools *interoperable*. The proposed *integration framework* consists of an *ontological model*, a *prediction module* and an *annotation module*. We have formalized the core concepts of

a SOAR platform that are discussed in chapter 3 in an ontology. The developed ontology is required to automate the execution of an IRP. We have followed a systematic approach to define the classes of the proposed ontology and the relationships among the classes.

We have designed and developed a prediction module utilizing the existing Natural Language Processing (NLP) and Machine Learning (ML) techniques to automatically classify the activities with text descriptions according to the ontology. For a new activity description in an IRP, we have performed a text-based similarity measure with the existing list of activities descriptions. We have defined a threshold for the similarity measure that is used to invoke the prediction module when the similarity score is above the threshold. For a similarity score below the threshold, we have designed an annotation module to generate and recommend the possible classes to experts and automatically annotate the new classes in the ontology after an expert selects the classes.

We have designed and implemented an interoperability model to select the best suite of tools that have the required capability to execute an IRP. We checked the compatibility of the set of selected tools for interoperability based on their capabilities in terms of their input, output and execution environment. In this chapter, we do not show the development and evaluation of the ontology; instead, we demonstrate the use of the ontology by the prediction module and interoperability model for auto-execution of IRPs. The development and evaluation of the ontology for security tool integration is shown in chapter 5. The following are the key contributions expected in this chapter:

- An ontological model to formalize the diverse activities and capabilities of security tools (refer to section 4.4.1).
- A prediction module to automatically classify activities according to the ontology and an annotation module to annotate the unmatched activities with the existing ontology (refer to section 4.4.2 & 4.4.3).
- An interoperability model to select the security tools to automate the sequence of activities in an IRP (refer to section 4.5).

4.2 RELATED WORK

A large-scale SOAR platform requires formalization of the concepts of different security tools and their respective activities. Most of the existing literature on SOAR only focuses

on providing APIs or plugins for multi-vendor tools, without considering the importance of formalizing the standard features or concepts used by different tools [20, 23 , 24, 35]. STIX, CyBox, and Unified Cybersecurity Ontologies (UCO) are examples of some of the known ontologies for the security domain. UCO combines the existing ontologies; however, it does not provide an ontology for security tools and their activities; nor does UCO support an IRP's activities, which are required by a SOAR. A few studies formalize various concepts of information security, threat and attack related information for sharing the information amongst the security community [171-173]. None of these studies focus on formalizing the concepts of IRPs or the diverse nature of security tools.

One recent study has developed ontologies for enabling tool-as-service (TSPACE) for cloud-platforms [158]. Based on the stakeholder's requirements and tools' artifacts, the required tools are selected using the ontologies, which helps the stakeholder to alleviate the semantic conflict while integrating multiple tools. The proposed ontology in TSPACE cannot automate the execution of the activities or enable interoperability among security tools. Moreover, TSPACE does not capture the capabilities of tools essential for *interpretability* and *interoperability*. Conversely, our proposed ontological model provides the capabilities of security tools to support *interpretability* and *interoperability* of security tools in a SOAR. Our work supports the *interoperability* issue by mapping the capabilities of the security tools with the activities of an IRP. Using the ontological model, a SOAR is able to *interpret* the diverse security tools' capabilities to make them work together to *automate* the execution of the security tools' activities without any human intervention.

Alongside the general lack of *interpretability* and *interoperability* among multi-vendor security tools, we could not find any work that addresses the issues with changing IRP due to emerging threat behaviors. Our proposed prediction module supports the auto-classification of new activity descriptions according to the ontology for automatic execution of IRP. To the best of our knowledge, this is the first work that has enabled *auto-integration* of security tools in a SOAR based on an ontology, and developed a prediction module to classify activity descriptions based on the ontology. The *automation* is achieved by enabling *interpretability* and *interoperability* among a variety of security

tools from different vendors and *auto-classification* of activity descriptions according to the ontology.

4.3 MOTIVATION SCENARIO

An incident is any unwanted event that violates specific security objectives (confidentiality, integrity, and availability) of an organization's assets. An IRP aims to provide the best sequence of activities to be performed in response to an incident, e.g., alerts for *phishing emails*, *DDoS attacks*, and so forth. Table 4.1 shows an IRP for one such incident, spear-phishing email. A phishing email is used to obtain sensitive information by disguising as a trustworthy entity in electronic communication.

Table 4.1. The incident response plan for a phishing attack

#	Response	Activity Description
ac_1	*Is this* a Phishing attack?	*Validate* if this is a phishing attack.
ac_2	*Scan* Endpoint – Malware found?	After running a *scan*, *determine* whether malware was found.
ac_3	*Remove* Malware – Success?	*Determine* whether the malware was successfully removed.
ac_4	*Wipe* and *reimage*	If you did not successfully *remove* the malware found, this task instructs you to perform a *wipe* and *reimage* of the infected computer
ac_5	*Update* email protection software	If it was determined as phishing attack, you are prompted to *update* the email protection software accordingly
ac_6	*Remove* unread phishing emails	Perform the steps necessary to *remove* unread phishing emails still in the queue.

Figure 4.1 shows a scenario for SOAR where it collects the details of an incident, checks in the playbook for the corresponding IRP and rules therein, selects the tools to perform the activities based on the rules, orchestrates the activities and automates the execution of an IRP. Most SOARs have a playbook, as shown in Figure 4.1, where a SOC defines the rules based on their respective IRPs. The SOAR shows the scan and ongoing operation through its dashboard, based on which a SOC team makes the required decisions, defines new rules in the playbook and performs complex analysis. We refer to the activities that are performed by SOAR to orchestrate and automate an IRP as *Tasks*.

To address the interoperability issue, an existing SOAR offers APIs or plugins to communicate with different security tools. Most of these APIs or plugins are not vendors' or tools specific and fail when updates or changes are required [20, 32, 35]. There are several challenges associated with existing SOAR; however, in this work, we only focus on the challenges mentioned below. We use the example of Table 4.1 to illustrate the challenges that arise during the auto-execution of IRP by SOAR platform.

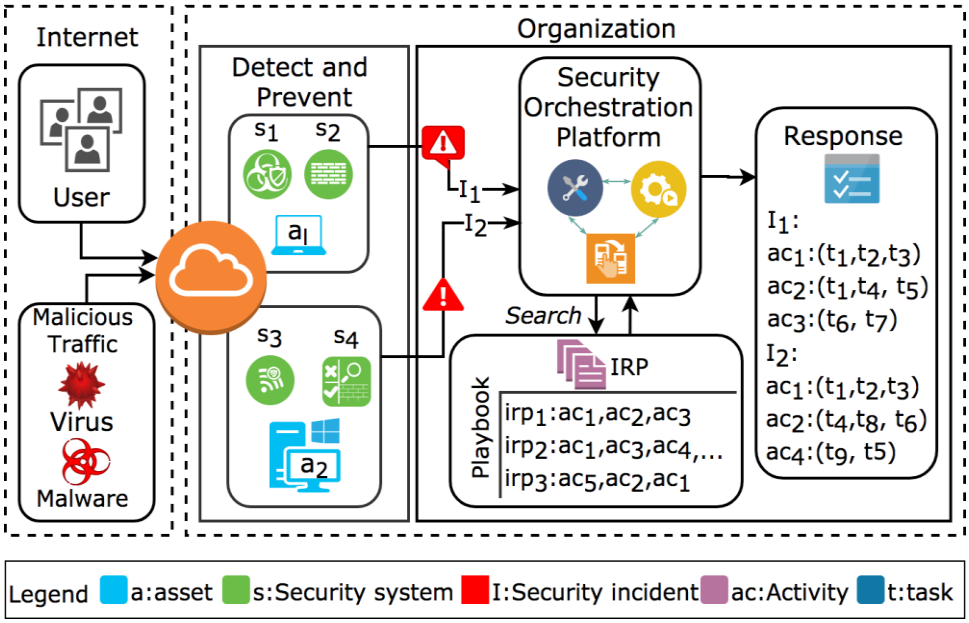


Figure 4.1 Overview of a security orchestration platform

Firstly, the IRP of Table 4.1 is written in text and does not follow a formal structure. There exists ambiguity among different words. Different words are used to define the same types of activities. For example, both *Response* and *Activity Description* in Table 4.1, i.e., "*Is this a Phishing attack?*" and "*Validate if this is a phishing attack*", are referring to the same activity. A SOC does not follow any specific structure while defining the activities of an IRP. The similar types of activities performed for different security incidents require different tools. For example, "*remove malware*" and "*remove phishing email*" both refer to the activity "*remove*", even though the execution of these activities requires two different types of security tools. A SOAR cannot automatically *interpret* the abovementioned similarities or ambiguity.

Secondly, a SOAR needs to deal with different tools that are not interoperable to automate the execution of an IRP's activities. For example, to execute an activity ac_1 of

Table 4.1, a threat intelligence platform, e.g., *Malware Information Sharing Platform (MISP),* is needed. A *MISP* is used by a SOAR to validate the incident. The execution of ac_2 requires an *EDR* tool to scan endpoints and *a SIEM* to identify the malware from *EDR* logs. Each activity has one or multiple rules associated with it. A SOAR uses these rules to orchestrate and automate an IRP by using different security tools. For example, if ac_1 is true, then only it executes ac_2. Based on the results of ac_2, it further executes ac_3 or other activities. shows an IRP for one such incident, a spear-phishing email. A phishing email is used to obtain sensitive information by disguising itself as a trustworthy entity in electronic communication.

Thirdly, a SOAR needs to control the flow of the activities performed by different tools. Experts modify the activities based on the tool's availability and preferences. For example, an expert may change one activity description in an IRP from "*analyzing the alert log*" to *"correlating alert log"* after installation of a new IDS in the network router. Installation of a new server requires the security tools' capabilities to fulfil the security requirements of a server. An IRP team defines the plan to protect the server from security incidents. In case existing tools are unable to provide the required capability, a SOC integrates new security tools to protect the server.

Fourthly, there may be multiple tools available for execution of a single activity. For example, different *EDR* tools and dedicated malware detection tools are used to perform "*scan endpoint for malware*." There is a lack of a systematic approach that can be followed to perform the selection of security tools that are interoperable.

In terms of changing activities in an IRP that needs integration of new tools, the challenge is how to provide an interoperability model for a variety of security tools to automatically execute different sets of IRPs. In the next sections, we first propose the semantic integration framework and then an interoperability model that uses the correct component of the integration framework to address the abovementioned challenges.

4.4 AN INTEGRATION FRAMEWORK FOR A SOAR PLATFORM

4.4.1 An Ontological Model to Enable Semantic Integration

A SOAR deals with various types of data produced by heterogeneous security tools. These data can be structured, semi-structured, or unstructured. Data produced by one tool

are not always interpretable by another tool. Therefore, these heterogeneous security tools are not *interoperable*. We develop an *ontological model* to represent multi-sourced data and enable semantic-based data integration among heterogeneous security tools in a SOAR [171, 172]. We define the classes of the required ontology by following a structured approach to keep consistency among the classes.

4.4.1.1 *Design and Development of an Ontology Class*

We follow a bottom-up approach to develop the main concepts of our ontology, which contains three main classes: *SecurityTool*, *Capability*, and *Activity.* These classes are defined to represent heterogenous security tools from different vendors formally. We leverage the TSPACE work [158] to design the capabilities of security tools in terms of their functional and non-functional features. The functional feature is the ability of a security tool to execute an activity such as *packet capturing*, *log management*, *intrusion detection* and so forth. The non-functional features include input and output data structures, and the configuration details required to execute an activity. For example, a network-based IDS takes *network traffic* or *packet* (i.e., *tcpdump*), where a host-based IDS works with *system logs (i.e., syslog)*. Even though both types of IDSs produce alerts as an output, the *output format* (*i.e., PCAP, CSV*) and data (e.g., *IP address, Port, MD5, and URL*) also vary, depending on the SOC's preferences.

The *Capability* class of the ontology consists of the two subclasses, *FunctionalCapability* and *NonFunctionalCapability,* to capture the features of security tools, as shown in Figure 4.2. The diversity among input and output data structures is apprehended using three subclasses under the *Non-FunctionalCapability* class: *Input*, *Output*, and *RuntimeEnvironment.* The input and output of the security tools need to be explicitly defined to be analyzed by a SOAR. A well-designed *Capability* class enables SOAR to auto-generate the APIs between security tools by retrieving the information about required input commands and produced output. The ability of a SOAR to deconstruct the output of one tool and then to use the output to formulate the input of another tool enables *interoperability* between isolated security tools.

We analyze the functional capabilities of multiple security tools to identify the subclasses of the *SecurityTool* class, where each tool has more than one functional capability. The *SecurityTool* class is categorized based on the main functionalities of the

security tools. We define the first level of the subclass of the security tool based on the types of activities (e.g., *detect*, *monitor*, *scan*, *validate* and so on) they provide. For example, *IDS*, *SIEM*, *Antivirus*, and *Firewall* are different types of security tools that are defined as a subclass of the *SecurityTool* class. The available commercial and open source security tools are categorized under each of these subclasses, based on the benchmark of their functional capabilities. For example, different types of SIEM, i.e., *Splunk* and *RSA NetWitness,* are subclasses of SIEM.

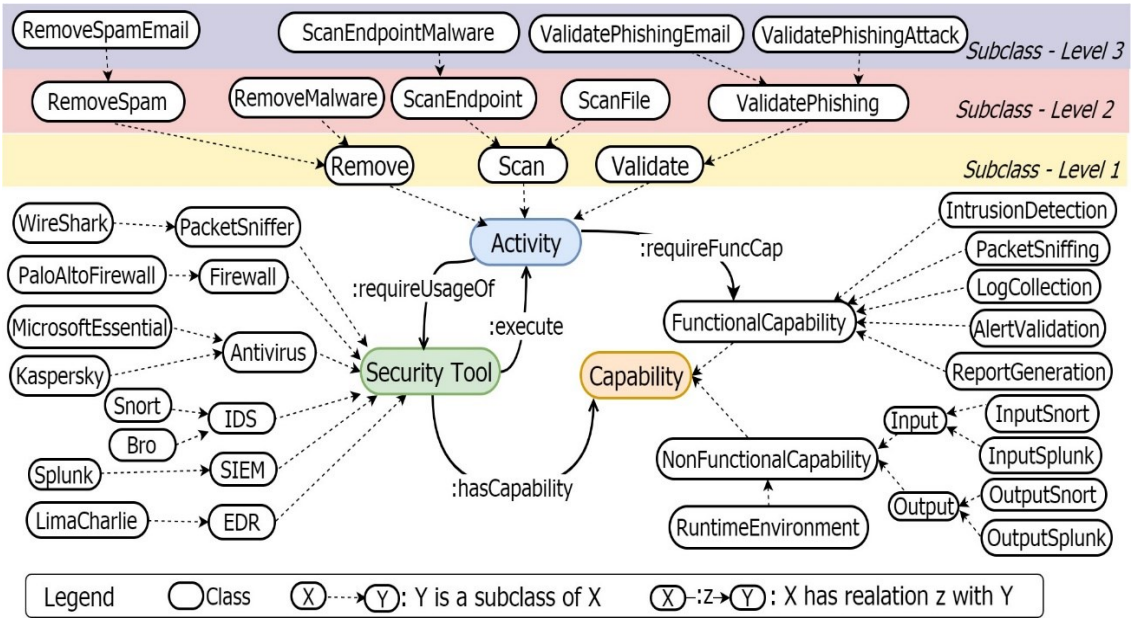


Figure 4.2 Excerpt of our Ontology

We define and categorize different types of activities as the subclass of the *Activity* class. The activities are associated with the detection, prevention, recovery and remediation actions of a threat defense and response life cycle. We follow a systematic set of guidelines to define the subclasses of the Activity class manually. First, we only use the verb and noun of the sentence of activity description to define the subclasses of the Activity class. For example, for the activities of Table 4.1, *Validate*, *Remove*, *Scan*, *Wipe*, *Reimage* and *Update* are the subclasses of level 1 of the Activity class. Then, we combine the adjacent verb, noun, and adjective and discard all other parts of speech to define the categories of the subclasses, as shown in Figure 4.3.

Each subclass of the Activity class contains multiple subclasses, based on the capabilities required to execute the activity. For example, the execution of two validation

activities: *validation of a phishing email* and *validation of exposure of confidential information* require different capabilities; therefore, they are categorized under different subclasses: *ValidatePhishingAttack* and *ValidateDataExposure*. We also consider the activity "*Is this a phishing attack?*" under the class *Validate*, as this is more similar to validating whether an alert/attack is phishing or not. We consider a different sentence with similar meaning in the same class. For example, the activity "*scan endpoint for malware*" and "*scan host for malware*" requires the same types of capabilities and thus they are categorized under the same class, *ScanEndpointMalware*. These subclasses can have more subclasses, depending on the requirements to execute the activities. Figure 4.2 shows part of the subclasses of the *Activity* class that we have built following the abovementioned process.

> ***ac_1 :*** *Is (**Verb**) this (**Det**) a (**Det**) phishing (**Verb**) Attack (**Noun**) ? (**Punc**) = Is (Validate) Phishing Attack*
> ***Subclass:*** *Validate → Validate Phishing → Validate Phishing Email*
> ***ac_2:*** *Determine (**Verb**) whether (**Adp**) the (**Det**) data (**Noun**) associated (**Verb**) with (**Adp**) this (**Det**)*
> *is (**Verb**) sensitive (**Adj**) = Determine Data Sensitivity*
> ***Subclass:*** *Determine → Determine Data → Determine Data Sensitivity*

Figure 4.3 The parts of speech tagging of the incident response plan and removing stop words

4.4.1.2 *Defining Relationships and Constraints.*

We define the relationship between the classes to select the tools with appropriate capabilities to execute an activity. The relationships between the classes are shown in Figure 4.2. We define a set of reasoning rules to enhance the relationships between different classes for error-free integration. These rules enable us to express conditions about the occurrence or non-occurrence of the required activities, the creation of instances, tracking and managing activities of a SOAR. For example, each security tool must have at least one functional capability associated with threat defense and incident response to execute an activity. The security tools must satisfy the capabilities associated with a class to be part of that class.

Execution of each activity depends on the availability of the relevant security tools and the preference of an organization's security requirements. Auto-execution of an

activity requires at least one tool with the required functional capability to execute a desired activity. We impose different types of restrictions for creating the instance of a class that must satisfy the relationship it holds with other classes. The defined rules enable a SOAR to avoid ambiguity, while creating an instance of a class. A SOAR executes the activities sequentially; as a result, the security tool that is selected to execute ac_{i+1} must have access to the output of a security tool that executes ac_i. For example, if *Splunk* is required to analyze the alert log produce by *Snort*, it must have access to the output file of *Snort*. Similarly, a SOAR needs to have the authorization to run and stop every security tool that is integrated into it.

The proposed ontological model enables a SOAR to interpret activities and security tools' capabilities. Retrieving the information of the non-functional capability class, SOAR can *interpret* the data generated in various forms and formulate the input command to invoke a tool for auto-execution of the activity.

4.4.2 Classification of Activities based on Text Similarity

A SOC adds new types of activities or updates the existing IRP to keep the playbook updated for emerging threats. Considering the tools available to execute IRPs, we leverage existing NLP and ML techniques to automatically classify the new activity description according to the activity ontology. This process makes the SOAR capable of analyzing an IRP and transforming the data into a representation that gives both an analyst and a machine insights about the data. We consider the classes of Activity classes in different levels separately (Figure 4.2). An example of a class on each level would include: level 1 {*Remove*, *Scan*, *Validate*}, level 2 {*RemoveSpam*, *RemoveMalware*, *ScanFile*}, and level 3 {*RemovePhishingEmail*, *ValidatePhishingEmail*}. From the perspective of ML, this problem is designed as a multiclass supervised text classification problem.

Given a new activity description in an IRP, we design the prediction module to classify the activity description according to the classes of the ontology. The overall workflow of building an ML-based prediction module is given in Figure 4.4. The dataset consists of the activity descriptions labelled according to the ontology. Table 4.2 shows examples of the labels that correspond to the activity described in each level of the ontology for each Activity class. Initially, the dataset is divided into training and testing

sets. The key components of building the ML model include text pre-processing, model selection, model building, and prediction. The model selection and model building processes work on the training set and the prediction process works on the testing set or with new activity descriptions.

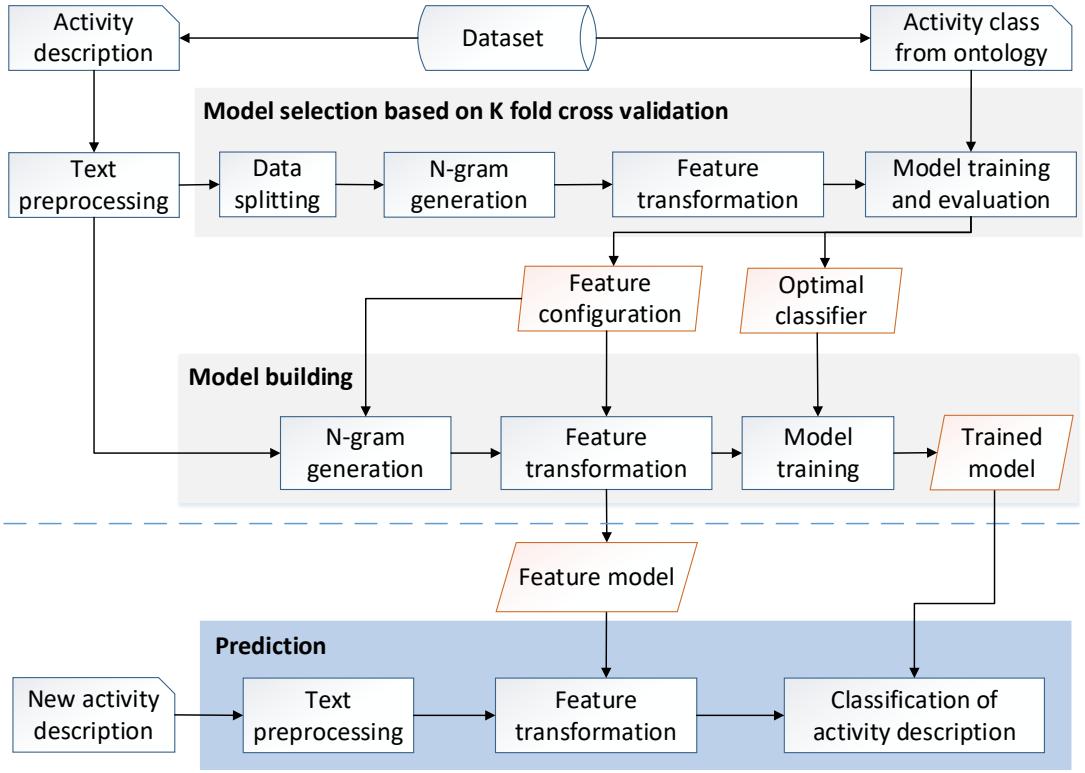


Figure 4.4 Development of the prediction module

Table 4.2. Activity description and corresponding class label

Activity Description	Level 1	Level 2	Level 3
Scan endpoint to see whether malware was found	Scan	ScanEndpoint	ScanEndpointMalware
Is this a phishing email	Validate	ValidatePhishing	ValidatePhishingEmail
Isolate the malicious node from the network	Isolate	IsolateMalicious	IsolateMaliciousNode

Text pre-processing: We start with a corpus of activity descriptions and follow the standard process of text wrangling and pre-processing. During the preprocessing step, we remove the null-value, punctuation, stop words, and meaningless words for the

analysis. We perform part-of-speech tagging of the text before removing the stop words and only retain the verbs, adjectives, and nouns.

Model Selection: We use the preprocessed text to perform *k-fold cross-validation* to select the optimal classifiers for the prediction module. As shown in Figure 4.4, the model selection method has four steps: *data splitting*, *n-gram generation*, *feature transformation* and *model training and evaluation*. The preprocessed text in each fold is split into the training and validation sets with equal sample sizes. We generate *word-based n-grams* for the training and validation sets that are merely the combinations of adjacent words of length n. We combine the n-gram with the Term Frequency-Inverse Document Frequency (TF-IDF) for each activity description.

The ML-based classifiers cannot directly process the text documents. Most of them expect numerical feature vectors of a fixed size, whereas the raw text documents are of variable lengths. The features generated from the n-gram are presented into *Document-Term Matrix (DTM),* where each row corresponds to an activity description and each column corresponds with a word in the term.

In the model training and evaluation steps, we train the four classifiers (*Random Forest*, *Linear Support Vector*, *Multinomial Model* of *Naïve Bayes*, and *Logistic Regression*) on the training set and then evaluate the model on the validation set using different evaluation metrics (*accuracy*, *recall*, *precision*, and *f1-score*). The classifier with the highest average cross-validation score is selected as an optimal classifier. The process is repeated for each level (levels 1, 2 and 3). The optimal classifiers and feature representations are returned for all three levels.

Model building: The model building process uses the whole set of pre-processed training sets to generate the word n-gram. Here, n-gram generation and feature transformation are based on the identified feature configurations for each level of class. The generated n-gram vocabularies are combined with the feature configurations to create the feature model. The feature model is saved to transform the data for future predictions. The extracted features are trained with the optimal classifiers returned in the model selection process to build the prediction model for each level.

Prediction: The prediction process is used for both testing the trained model and classifying the new activity descriptions. In this process, the activity descriptions are first preprocessed and then, using the saved feature model, transformed into a feature set. Finally, the features set is used by the saved trained model to determine the class of the activity descriptions in terms of the ontology for each level. The prediction module reduces the manual analysis of the activity description by classifying the activities according to the ontology.

4.4.3 Design and Development of the Annotation Module

A new activity description may not always fall in any of the existing activity classes of the ontology. In this context, we are considering these types of description as outliers. To identify the outlier descriptions, we perform text-based similarity checking of the updated or new description with the existing activity description and measure the cosine similarity. We define a threshold for considering whether the description is an outlier in terms of the existing set of activity descriptions. If the new description is not an outlier, then only the description is sent to the prediction module. If the new description is considered as an outlier, we develop the annotation module to automate the generation of the possible list of classes, following the same set of guidelines proposed to design the *Activity* class in section 4.4.1. The generated classes are matched with the existing set of classes, and if none of the classes are found in the ontology, the annotation module recommends a possible list of classes to the user. Once the user selects the corresponding classes, it creates new classes for the activity description and, if required, requests additional details about the classes from the user to keep the ontology consistent.

4.5 INTEROPERABILITY MODEL FOR EXECUTION OF IRP

A SOAR may need to invoke a different set of security tools in a different order to execute a variable sequence of IRPs. For example, one IRP may include an activity *scan endpoint*, followed by another activity *correlate alerts log*, whereas another IRP may include *correlating alerts logs* followed by *scan endpoint.* Both of these IRPs require the same security tools in different orders. We provide the interoperability model for auto-execution of the required IRPs, whereby one tool can understand the output of other tools. The model also helps SOAR to interpret the output and input of different security tools.

For example, a SIEM tool needs an output of alerts produced by IDS and a system log produced by EDR to perform correlations. Figure 4.5 shows the overall workflow of the interoperability model, starting from gathering a security incident to notifying a SOC.

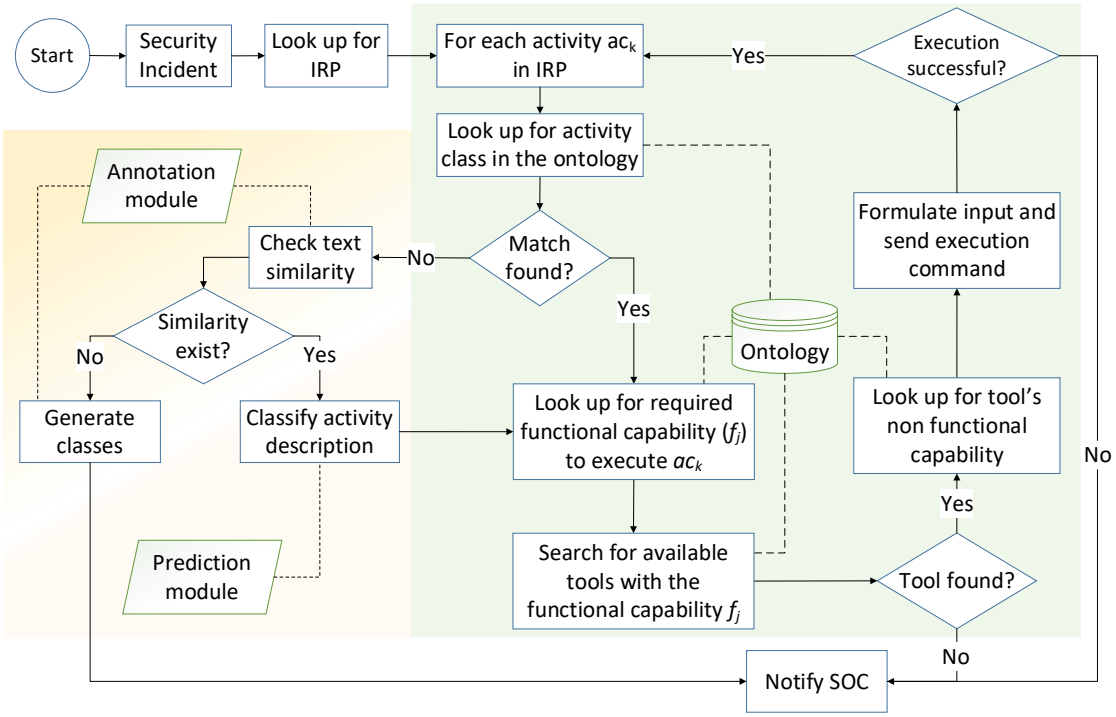


Figure 4.5 Workflow of the proposed solution

Two key tasks of the interoperability model are: select the desired tools based on their functional and non-functional capabilities and invoke the tools to execute an IRP. The key components of the integration framework (*ontological model*, *prediction* and *annotation module*), as shown in Figure 4.5, are used to design the interoperability model.

We have designed a Query Engine (QE) to retrieve information from the ontology. Given a set of Security tools $S = \{s_1, s_2, \ldots, s_m, \ldots\}$, a list of the required activities $AC = \{ac_1, ac_2, \ldots, ac_k, \ldots\}$ and a list of capabilities, $F = \{f_1, f_2, \ldots, f_j, \ldots\}$, a SOAR looks up the corresponding IRP for each security incident. For each activity ac_k of *IRP*, SOAR invokes the QE to search for the corresponding *Activity* class. If the activity is found in the ontology, the SOAR invokes QE to retrieve the capability required to execute the activity. Considering f_j is the required functional capability, a SOAR sends queries to retrieve the security tool that has the functional capability, f_j. In cases where multiple tools are available, the SOAR selects the right tool from the list. In the next step, the

SOAR retrieves the non-functional capability of the selected security tool to formulate the input command for instructing the tool to execute the activity. The QE extracts the necessary information from the ontology to formulate the input for the tool. After constructing the input command, the SOAR calls the tool's corresponding routine to execute the activities. If the execution is successful, the next activity in the IRP is executed by following the same sequence of tasks (performed by SOAR).

Considering the output produced by one tool, $s_{m,}$ is provided as an input to another tool, s_p, the SOAR checks for the interoperability of the two security tools. The SOAR deconstructs the output of s_m to formulates the input of the s_p. This action is only possible if the tools are interoperable; otherwise, the SOAR notifies the SOC. An activity's description may change continuously; if no class is found for a particular activity, a SOAR first invokes the AU unit to determine the possibility that new description is part of the existing ontology. Based on the similarity measurement, it either generates the list of classes or invokes the prediction module to classify the activity description. After receiving the appropriate class from the prediction module, the same steps of looking for the required functional capability and non-functional capability to execute the activity are carried out.

Following the abovementioned process, a SOAR can automate the sequence of activities in an IRP even when changes occur in the underlying execution environment. The interoperability model enhances the capability of a SOAR to automate the execution of an IRP by interpreting the activity, required capability, and tools' interoperability.

4.6 EXPERIMENTS AND RESULTS

We carried out a set of experiments to assess the feasibility of the proposed prediction module and the interoperability model. The flexibility of the proposed ontology-based sematic integration is discussed in chapter 5.

4.6.1 Preparing the dataset for a prediction module

Our experimental dataset is based on the IRP crawled from the website of *ServiceNow,* which resulted in 1080 activity descriptions. For each activity description, we labeled the classes manually, according to the ontology, as shown in Table 4.2. We have *34 categories* under level 1, *67 categories* under level 2 and *74 categories* under level 3.

4.6.2 Implementing the prediction module

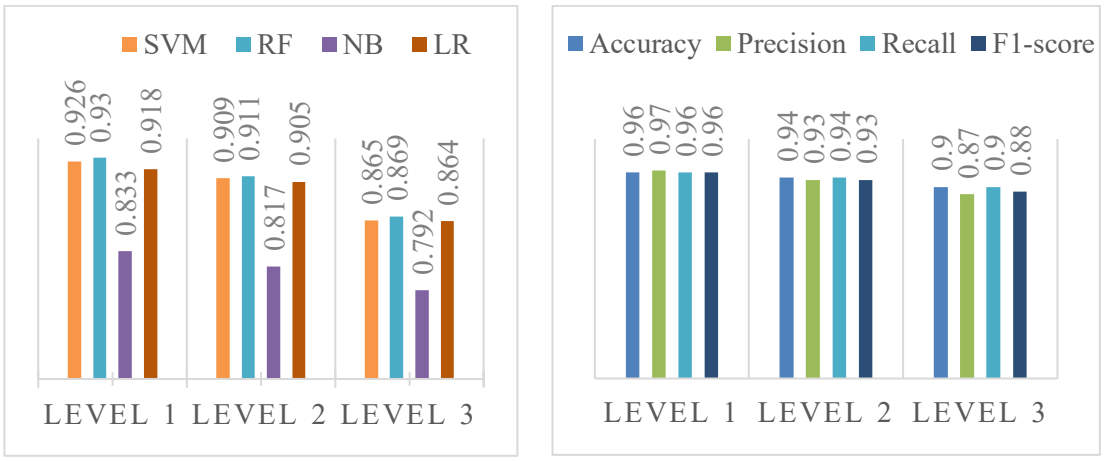
We used the *scikit-learn*, *NLTK* and *spaCy* packages of python to build a classifier. For each level, we first implemented four classification algorithms with different hyper-parameter settings separately. We performed k-fold cross validation for each configuration by splitting the data set into different training and validation sets. We used the function *GridSearchCV()* to select the optimal configuration and perform cross-validation for each classifier. For both *Support Vector Machine* (SVM) and Linear Regression (LR), we considered different values for the regularization parameter (i.e., 0.01., 0.1, 1, 10, and 100). For Multinomial Naïve Bayes (NB), we considered the prior probability of the class True and False. For Random Forest (RF), we considered different values for estimators (i.e., 10, 100, 20, 200, 50, and 500) and the maximum number of leaves (i.e., 10, 50, 100, and 200).

Figure 4.6 (a) shows the results of different classifiers for the optimal configuration. We examined the performance of the classifiers in terms of accuracy and F1-score [174]. An F1 score is considered more reliable than accuracy. *Accuracy* reflects the total of the correct predictions divided by the total number of cases. The F-1 score is the harmonic mean of Precision and Recall. The *precision* represents the total of the correct predictions for each class, divided by the total number of activities predicted for that class. The *recall* is the correct prediction for each category, divided by the total number that belongs to this category. Comparing the results of the classifier, we found that RF outperformed other classifiers. We built the final model with the RF classifier. The optimal configuration for *RF* (estimators, maximum leaf) for levels 1, 2 and 3 are (50, 100), (100, 100) and (10, 200), respectively. We used 70% of the activities for each level as the training data and 30% as the testing data. Figure 4.6 (b) shows the results of the RF for different evaluation metrics.

4.6.3 Developing the Interoperability model

We implemented a Proof of Concept (POC) system using seven security tools (Snort, Splunk, LimaCharlie*, Wireshark*, WinPcap, Microsoft essential, and MISP) to study the viability of the interoperability model. We described their capabilities in terms of the ontological model and used a list of IRPs with different activities. We used the network traffic and system logs as the input to identify the security incidents. The experimental

study used 21 different capabilities and 9 IRPs with 17 activities. We only considered the activities for which the capabilities were available. We changed the activities and observed the corresponding changes in the operation's execution.



(a) Validated Weighted average of F1-Score (b) Testing results of the Random Forest classifier

Figure 4.6 Bar plot of (a) validated weighted average of the F1-score for optimal configuration of different classifiers and (b) testing results of Random Forest for three levels of class

Discussion: The results showed that in more than 90% of cases (Figure 4.6 (b)) the prediction module classified the activity descriptions accurately. The performance in classifying the activities in levels 2 and 3 is lower than that in level 1. The reason for this appears to be the number of members in these classes is lower than that in level 1. The more input data we can provide to the classifier, the more accurate results it will produce. Furthermore, the activity description was passed to the prediction module only when the text similarity was found, which makes the classifiers less error-prone towards the new activity description that does not belong to any of the existing classes. Out of the 17 IRPs, the POC was able to automate 15 IRPs successfully and 2 IRPs partially. While modifying the activity descriptions, there were two activities (update email protection software and detect phishing email) for which security tools were not available. For these two activities, the interoperability model was unable to find suitable security tools, thus failed to automate the execution of that particular IRP. Except for these two activities, the POC automatically (a) retrieved the information from the developed ontology; (b) generated the configuration details to call the desired security tools; and (c) enabled *interpretability* and *interoperability* among the different security tools and SOAR.

4.6.4 Threats to validity

We developed the ontology based on freely available and open source security tools' capabilities, and activity descriptions, which might not fully represent the situations or scenarios of an organization. Considering the development of an ontology is an incremental process, a human expert can easily extend the ontology to incorporate the tools used in an organization. The selected optimal may not guarantee the highest performance for classifying the new and updated activity descriptions since an infinite number of configurations are available to tune the hyper-parameters of ML classifiers. The selected classifiers might not be the best, but the system provides a learning-based approach to classify any activity description, which can be further improved and extended with different classifiers and configurations. The model we built is retrainable and can be easily trained with a new dataset.

4.7 CHAPTER SUMMARY

Given the widespread adoption of SOAR over the last couple of years, there is an increasing demand for self-adaptive SOARs. Our research seeks to devise a solution that can enhance the *integrability*, *interpretability* and *interoperability* of security tools integrated into a SOAR. The proposed approach allows a SOAR to select the required security tools that are interoperable for auto-execution of an IRP. We have introduced an ontological model to formalize the security tools, their capabilities, and the activities of an IRP. A learning-based prediction module is proposed to reduce the manual work of a security team to define the classes for activity in a playbook. The proposed interoperability model successfully automates the execution of most of the IRPs at runtime. In future work, we will extend the system to automate the generation of the APIs from the ontology. We also aim to use the semantic definition of tools' capabilities to auto-create the APIs when new security tools with new capabilities are integrated, and design a probabilistic model for selecting and integrating security tools.

Chapter 5

An Ontology-driven Integration of Security Tools

As discussed in Chapter 4, the lack of interpretability and interoperability among security tools is considered a key challenge to fully leveraging the potential of the collective capabilities of different security tools. The processes of integrating security tools are repetitive, time-consuming and error-prone; these processes are carried out manually by human experts or using ad-hoc methods. To help automate security tools' integration processes, in this chapter, we propose an Ontology-driven approach for a SOAR platform (OnSOAR). The developed solution enables *interpretability* and *interoperability* among security tools, which may exist in operational silos. We demonstrate OnSOAR's support for automated integration of security tools to execute the incident response process with three security tools (Splunk, Limacharlie and Snort) for a Distributed Denial of Service (DDoS) attack. The evaluation results show that OnSOAR enables a SOAR platform to interpret the input and output of different security tools, produce error-free integration details, and make security tools interoperable with each other to automate and accelerate an incident response process.

5.1 INTRODUCTION

A SOAR platform aims to minimize dependency on human experts for a streamlined incident response process [70, 175]. However, most of the existing SOAR platforms cannot automatically adapt to changes in organizational systems' operational processes, such as the installation of new software, deployment of new servers, and rolling out of new access control policies [21, 22, 72]. Security teams need to integrate different

security tools with a SOAR platform manually and map their activities into an incident response process [18, 118, 176].

Human-centric intervention is required in the existing SOAR platforms because security tools are not designed to interoperate with each other [168]; for example, an IDS cannot automatically send an output alert to a SIEM. The messages generated by a security tool are also not semantically interpretable [35, 177]; for example, an IDS may generate an alert in its proprietary format, and such an alert may contain different features of an attack. A SIEM cannot ingest and interpret the meaning of those alerts unless the alerts' definitions are explicitly defined for a particular type of SIEM [35, 83]. Chapter 1 provides an overview of a SOAR platform that shows that, before deploying a SOAR platform, an organization assesses the existing security tools to identify the configuration details, such as dependency among different activities, process flow within a security tool, input and output data formats and runtime environments [35, 176]. Based on such an assessment, a SOAR is designed, and different security tools are integrated using plugins or APIs [21, 118, 176, 178].

Human intervention can be minimized by providing SOAR platforms with a formal specification of the security data format, configuration, and structural specifications of security tools to automate the process of integrating different security tools. A SOAR can use such formal specifications to continuously integrate and invoke security tools, based on the activities in the incident response process. Ontologies can be used to provide the required formal specifications to support semantic integration [172, 179-181]. In the context of SOAR, semantic integration means one security tool can understand the semantics of the input and output of another security tool. A SOAR can semantically interpret the activities when the formalization incorporates semantic integration of heterogeneous security tools.

Several studies have developed ontologies to formalize heterogeneous threat intelligence information for cybersecurity tools [172, 179-182], including ontologies to help stakeholders deal with any semantic conflict that arises while integrating multiple security tools [158]. These approaches focus on providing an effective method of information sharing and exchanging among cybersecurity communities, and stakeholders. None of these approaches provides support for sharing information

between different security tools and the SOAR. Security tools (e.g., IDS, SIEM, or EDR) are software-intensive systems that can be integrated and interoperate at the software level, based on data integration. Hence, ontological approaches can be leveraged to automate the process of integrating and interoperating heterogeneous security tools in a SOAR platform.

We have developed an **On**tology-driven approach for a **SOAR** platform, ***OnSOAR,*** to automate the process of integration of security tools. At the core of *OnSOAR* is an ontological model that involves both high-level and fine-grained classes for different security tools, their capabilities and the activities of an incident response process. The developed ontology provides a formal specification of the core concepts of a SOAR, i.e., security tools, their capabilities and the activities of the incident response processes. Based on the ontology, *OnSOAR* automatically annotates a set of security tools, their capabilities and an incident response process at a much finer-grained scale. We have also designed a set of *queries*, *rules,* and *constraints* for the orchestration process that enables *OnSOAR* to extract the required information from the ontology and invoke the required functionalities of a security tool. *OnSOAR* ensures *error-free* and *automated integration* of different security tools in a SOAR.

We have developed and evaluated *OnSOAR* with a robust incident response application. As a use case scenario, we have investigated an incident response plan for a *Distributed Denial of Service attack* (*DDoS*) with three different security tools. This can be extended for any other use case scenario and security tools. By composing a set of simple rules and defining structured queries for the orchestration process, we have shown that the developed *OnSOAR* can automatically select and invoke an appropriate set of functionalities from the available security tools. We have also shown that the developed *OnSOAR* can provide automation support to the process of integrating and interoperating security tools. The following are the main contributions of this chapter:

- Design and implementation of an ontology-driven approach, *OnSOAR,* that supports the process of automated integration and interpretation of a variety of security tools (Section 5.3, 5.4, 5.5, 5.6).
- Demonstration of the use of *OnSOAR* to automate the execution of an incident response process for a DDoS attack with three security tools (Section 5.7).

- Evaluation of *OnSOAR's* ability to automatically generate accurate configuration details for enabling security tools to *interoperate* and remove *operational silos* for a *DDoS* attack (Section 5.8).

5.2 PRELIMINARIES

5.2.1 Challenges in Automated Integration

Given the diverse nature of security tools, the integration process of SOAR has several challenges [183]. It is not possible to know all the requirements of an organization at the design and installation phases of a SOAR platform [171]. A SOAR platform needs to control the flow of the activities performed by different security tools. A security expert modifies the activities based on a system's availability and preferences. For example, a security expert may change an activity in the incident response process from analyzing alert log to correlating alert log after installation of a new IDS in a network router. However, there is a lack of a systematic way of automating the process of integrating security tools [22, 158].

A security expert adds or removes security tools or changes their configuration and deployment strategies; for example, a security team may change the preferred format of an alerts log file based on the SIEM system being used. A security expert manually maps the newly-integrated security tool's functionalities with the activities of an incident response process and vice versa. Figure 5.1 depicts a scenario of an organization that has a SOAR platform (i.e., similar to Figure 4.1 of chapter 4). However, the SOAR platform shown in Figure 5.1 cannot automatically perform the mentioned changes due to the heterogeneity and isolated nature of most security tools, which lack interoperability and interpretation of the generated messages.

5.2.2 Problem Formulation

Consider a scenario where an application of an organizations' website is being overused with a superfluous request. Detecting the scenario as a malicious one, an IDS generates alerts that consider the behaviors as a *DDoS* attack. Upon gathering such alerts, the SOAR orchestrates and automates the incident response process to prevent and recover from the *DDoS* attack. For illustration purposes, we assume the actions automated by a SOAR are *detecting incidents*, *collecting alerts* and *system logs*,

identifying the affected systems, and *generating an incident report*. Let us assume that an organization has a set of security tools S, that are integrated into a SOAR, where $S = \{s_1, s_2, s_3, \ldots, s_i, \ldots, s_n\}$. An example of s_i can be *Snort*, *Bro*, *Splunk*, or *Limacharlie*. Each security tool, $s_{i,}$ performs a set of activities, $AC = \{ac_1, ac_2, \ldots, ac_j \ldots, ac_m\}$ to protect against potential security attacks.

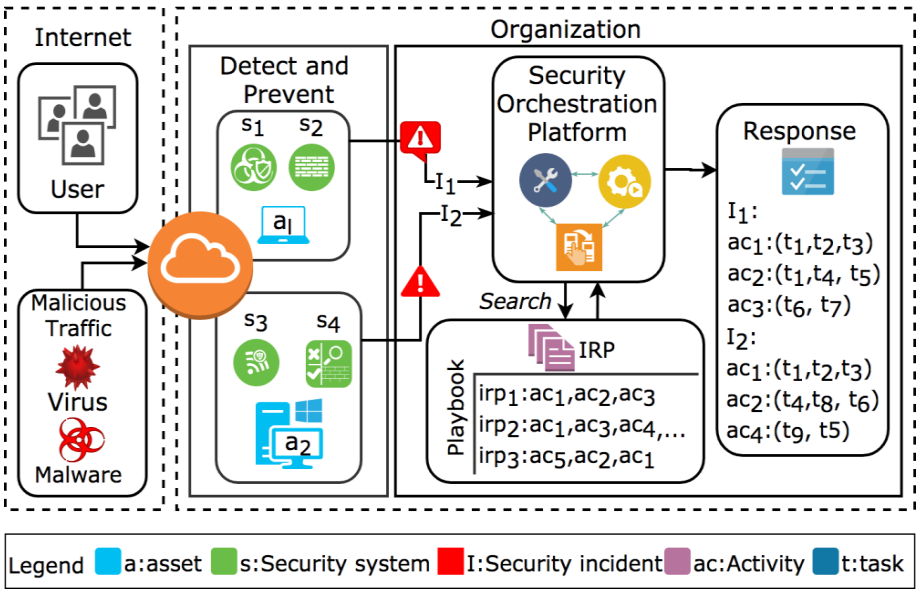


Figure 5.1 An example of execution of an incident response process in a security orchestration platform

Definition: 5.1. (Activity). An Activity is an action performed by a security tool or a human expert to detect, prevent, remediate, recover or respond to security incidents. Examples of the activities include *detecting security incidents*, *investigating threats,* and *analyzing threat behaviors*. A SOAR has a playbook that has a list of Incident Response Plans (IRP), $IRP = \{irp_1, irp_2, \ldots, irp_k, \ldots, irp_p\}$, where, $irp_k = \{ac_1, ac_2 \ldots, ac_j\}$ is a sequence of activities that needs to be executed in response to a security incident. The playbook contains the rules associated with the execution of an IRP.

Definition: 5.2. (Security Incident). A security incident is an unwanted or unexpected event/events that has a significant probability of compromising the security of an organization's assets. Examples of security incidents include security threats, breaches, attacks and so forth. For example, for a security incident, I $(DDoS\ Attack)$, the response process includes activities: *DetectIncident*, *CollectLog*, *AnalyzeLog*, *GenerateReport*, and so on.

Figure 5.1 shows a scenario for auto-execution of an incident response plan irp_k to respond to a security incident. In the scenario, a security tool, $s_{i,}$ detects suspicious traffic *on* asset a_l and generates an alert that it is considered as a security incident *I* (i.e., an alert "*DDoS attack*").

Definition: 5.3 (Asset): An asset is any system, data, resources, hardware, or software that an organization wants to protect. Examples of assets include databases, servers, hosts, applications, and websites.

A SOAR gathers the incident details and searches for an appropriate IRP in the playbook. For example, for the incident, *I*, shown in Figure 5.1, a SOAR finds the best match IRP irp_1, which has a list of activities $irp_1 = \{ac_1, ac_2, ac_3\}$. The SOAR then searches for a security tool that can execute those activities and finds the artifacts required to automate the execution of the activities. Then SOAR performs a set of tasks, *T* for each activity, where $T = \{ t_1, t_2, \dots, t_q, \dots, t_r\}$.

Definition: 5.4 (Artifact): We consider the alerts and logs generated by different security tools as the *artifacts* of a SOAR, which deal with different structured, semi-structured and unstructured data that come in various formats and languages from different security tools.

Definition: 5.5 (Task). A task is an action that a SOAR performs to automate the execution of the activities in an IRP. For example, the execution of an activity *DetectMaliciousTraffic* requires a SOAR to perform three tasks: *searchDetectionSystem* (t_1): looking for an available security tool that purports to detect intrusion, *selectDetectionSystem* (t_2): if multiple systems are available, selecting one, and *invokeDetectionSystem* (t_3): invoking a security tool to run in detection mode.

We assume that a SOAR selects a security tool s_i *(Snort*), which can scan the asset a_l (*endpoint*) to detect incidents. A SOAR invokes s_i to run in intrusion detection mode. These tasks are performed by a SOAR to automate the execution of the activity ac_j (*DetectIncident*). Finally, s_i scans a_l to detect suspicious traffic. We refer to this as the execution of the activity ac_j, that is performed by a security tool s_i. We consider the combined activities performed by security tools and the tasks performed by SOAR to automate the execution of activities to be an automated integration process.

5.2.3 Motivation

Our research aims to automate the process of integrating different security tools into a SOAR platform. The integration process is a combination of interpretation, selection, formulation, and invocation. A SOAR platform performs different types of tasks to manage different aspects of threat defense and incident response. For example, a SOAR connects different activities of different security tools to *remove operational silos*. We refer to this type of task as an *integration* task. A SOAR also orchestrates the flow of data and activities to make security tools *interoperable* and enables machine to machine automation by providing machine *interpretability semantics*. We consider these types of tasks as *orchestration*, and *automation* of security activities. The process of automated integration of security tools for the execution of an IRP depends on the combination of these three tasks.

Auto-Integration Process = task (Integration, Orchestration, Automation)
= Interpretation + Selection + Formulation + Invocation + Execution

In this work, we focus on developing suitable support for automating the process of integrating security tools in a SOAR for a seamless incident response without human effort in performing activities.

5.3 THE PROPOSED SOLUTION

Figure 5.2 provides an overview of the developed *OnSOAR*. It mainly comprises three layers: a semantic layer, data processing layer and security tool layer. It is composed of core concepts of a SOAR's platform and provides (i) the unified capability of a SOAR platform, which can be achieved through a number of security tools, (ii) the semantics of different security tools' generated data and (iii) the rules to avoid conflict and invalid integration of security tools. It can make different security tools interoperable, so that the output of one security tool can be used as input to another security tool without human intervention.

Security Tool Layer (Sec 5.5): The security tool layer provides the fine-grained information about security tools, running processes and the current state of an organization. In this layer, the artifacts generated by different security tools and assets are forwarded to the upper layers (Figure 5.2), which then extract the features (e.g., alert

types) from logs. Figure 5.2 shows that the raw data are passed through the integration layer to the data processing layer, which applies pre-processing rules to map the raw events onto the classes of the developed ontology (e.g., maps the alerts log to the IDS that generated it). The integration layer mainly consists of APIs, plugins and wrappers that are used to integrate security tools in a SOAR platform (chapter 3). As the focus of this chapter is on automating the process for integrating the security tools, we do not discuss the integration layers at this point.

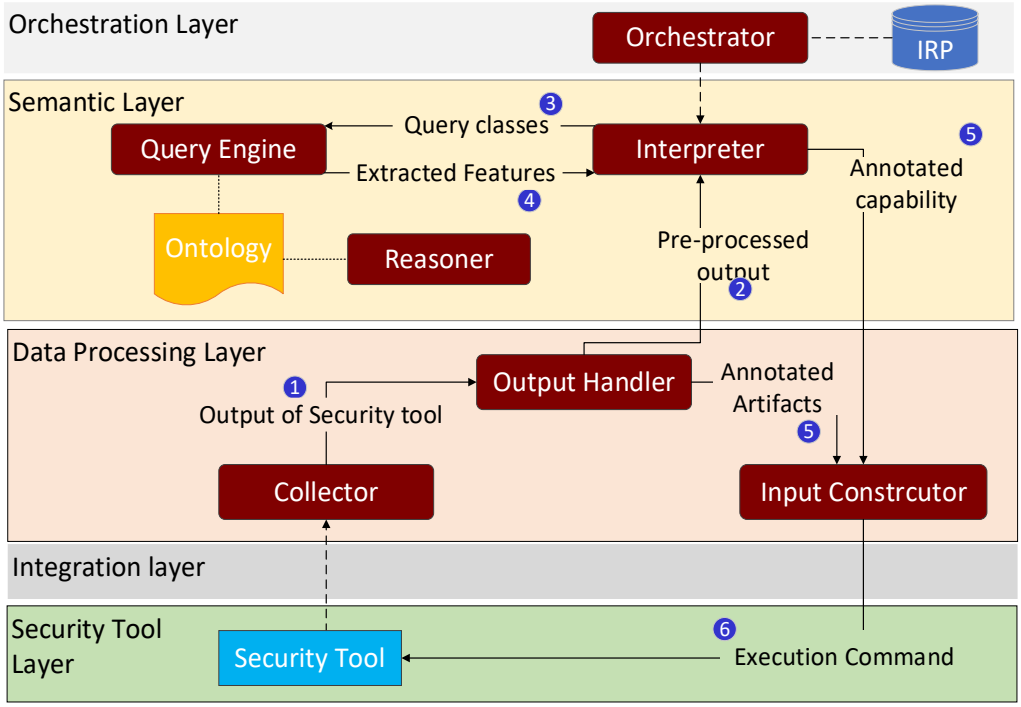


Figure 5.2 A high-level overview of OnSOAR

Semantic Layer (***Sec 5.4***): The semantic layer provides the semantic details about the input, output, and activities performed by security tools to the data processing layer and orchestration layer. It supports the integration process of the security tools to *OnSOAR* through which (i) security tools' capabilities (functional and non-functional features) are captured, (ii) the capabilities required by IRPs are identified, (iii) the related artifacts and data maintained among different security tools are identified and (iv) the configuration details of security tools are retrieved. It also stores abstract knowledge about the tasks performed by SOAR.

Data Processing Layer (Sec 5.6): The data processing layer is responsible for collecting and analyzing security tools' data and invoking security tools. The collector

collects the system logs and output of security tools. The output produced by different security tools are passed to the interpreter. Furthermore, the annotated artifacts from the semantic layer are passed to the data processing layers' input constructor to formulate commands for invoking a security tool.

The orchestrator of the orchestration layer mainly controls the integration process. The orchestration layer is responsible for invoking appropriate tasks for automating the process of integration of several security tools based on the activities of an IRP (details in chapter 3).

5.4 SEMANTIC LAYER

Interpretation of security tools' capabilities requires formalization of various concepts (i.e., activities, inputs and outputs). The semantic layer uses an ontology to represent the domain knowledge of the SOAR platform through a set of concepts and their relationships, as described in the following section. It leverages the ontologies' capability to represent multi-sourced data [179, 182], taking into account the semantic integration process among heterogeneous data produced by different security software systems.

5.4.1 Ontological Model

To build the ontological model for *OnSOAR*, we have cataloged the existing security tools based on their key features, different data types and runtime environments. Our ontology engineering focuses on leveraging existing and widely-adopted ontologies [158]. We used the ontological model to formalize the semantics of some of the key security tools' capabilities (e.g., *intrusion detection* and the *command to invoke a system*), artifacts (i.e., windows log, Syslog) or context data (affected assets) and the activities of IRPs. The proposed ontology model consists of the following classes, shown in Figure 5.3. The *security tool class* (*SecurityTool*) represents all types of security tools. These systems are designed to protect assets, based on an organization's requirements. We modeled different features of security tools under this ontology class. The *activity class* (*Activity*) formalizes the actions of the IRP. The activities are presented with respect to a system's required functional features. The *capability class* (*Capability*) is used to capture functional and non-functional features of an individual security tool. This class is used to instantiate the underlying ontology model in response to a security incident to

execute the selected activities. It also formalizes the types of data with which security tools deal. As shown in Figure 5.3, the *Activity* class has a relation *requireUseOf* with *SecurityTool* class, where the *SecurityTool* class has a relation hasCapability with the class *Capability*. The relationship between classes is defined as the object property. Here the hasCapability is the object property of the *SecurityTool* class. These types of relationships can also be presented as a triplet in RDF or XML, for example, (*SecurityTool*, hasCapability, *Capability*) *w*here *SecurityTool* is the domain of the object property and *Capability* is the range.

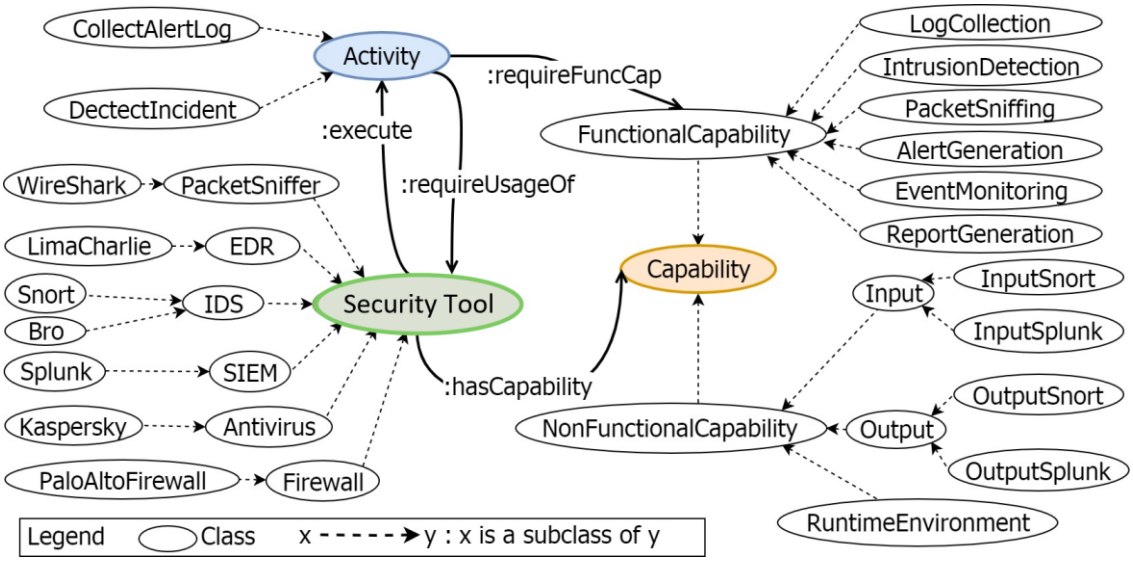


Figure 5.3 Part of our ontology: the dashed arrow represents the subclass and the solid line represents the relationship among classes

5.4.1.1 ***Security Tools***

Different types of security tool have been categorized under the abstract class *SecurityTool*, which has different subclasses, with some subclasses based on their functionality, as shown in Figure 5.3. For example, both *Bro* and *Snort* in Figure 5.3 are considered as the subclass of *IDS* due to their extensive use of intrusion detection. Though *Snort* can be used for both intrusion detection and packet sniffing, most organizations use *Snort* to detect intrusions, which is the main functionality of *Snort*.

We express the necessary conditions for a security tool to execute an activity through the semantics of the *Capability* class. The *SecurityTool* class can be extended to incorporate new security tools with new types of behaviors. When *OnSOAR* intends to integrate a security tool, it creates an instance of a security tool by instantiating the

associated properties required for the security tool's integration. For example, an instance of *SecurityTool, SnortInstance* represents the instance of *Snort* class, which indicates a *Snort* system is integrated into *OnSOAR*. To categorize a security tool under a particular subclass, the security tool must satisfy the capability associated with that subclass. We impose different types of restrictions for creating an instance of a class that must satisfy the relations with other classes. For example, as shown in Figure 5.3, the *SecurityTool* class has a relation *execute* with the *Activity* class. An instance of a *SecurityTool* class must execute an activity. We describe more details on the rules for imposing different constraints and restrictions in section 5.4.1.

5.4.1.2 ***Activity***

We categorize each activity of the IRP under the *Activity* class. This class is instantiated in response to an incident I (i.e., *DDOS* attack). An instance ac_i (i. e., *DetectMaliciousTraffic*) of an activity class ac_j, $ac_j \in AC$ represents the execution of ac_j in response to the incident *I*. *DetectMaliciousTraffic* and *CollectSnortAlertLog* are instances of the subclass of the *Activity* class. Figure 5.3 shows the details of the subclasses and their relationships. Execution of each activity further depends on the availability of security tools and the preferences of an organization's security requirements.

A SOAR executes an orchestration routine to call individual security tools to execute an activity. Execution of activities generates artifacts, i.e., system and alert logs. Artifacts are also required before executing the activities. For each activity, a SOAR performs a set of tasks T (i.e., *select security tool* and *invoke security tool*) to collect and manage artifacts, automate, and track execution of the activities performed by the security tools. The execution of these tasks generates further events, i.e., *system found* and *endpoint protection system running successfully,* through which *OnSOAR* keeps track of the tasks and the activities being executed by a security tool.

5.4.1.3 ***Capability***

We define the capability of a security tool under the class *Capability* with two subclasses: *FunctionalCapability* and *NonFunctionalCapability*. We consider each security tool can be represented in terms of their functional and non-functional capability.

Definition 4.1. (Functional Capability). The functional capability is the capability of a security tool to perform activities to achieve security objectives. We denote a set $\mathbb{F}$ as the functional capability of a security tool. Each security tool, s_i, can have a list of functions denoted by δ_{s_i} where $\delta_{s_i} \subset \mathbb{F}$. For example, *IntrusionDetection* and *LogManagement* support the activities *DetectIncident* and *ManageLog,* respectively.

Definition 4.2. (Non-Functional Capability). The non-functional capability is the ability of a security tool to support the quality requirement, while providing the functional features. Examples of a security tool's non-functional capabilities include the command syntax, input parameter format, and data type. For instance, alerts generated by *Snort* in various file formats (i.e., *CSV* or *binary*) are considered as the non-functional capability.

Each security tool has a different data structure, preferred configuration, generated workflow and a way to share information with security experts [21-23, 72]. We define these kinds of the knowledge required by OnSOAR to run and maintain a security tool under non-functional capability. The Non-Functional capability class has three further subclasses: Input, Output, and RuntimeEnvironment. The proposed ontology model requires the Input class for executing an activity. The *SnortInputForIntrusio*n has the information to run Snort on *IntrusionDetection* mode. The Input class has the configuration details for Snort in intrusion detection mode. We have designed the Output class to capture different types of outputs that are generated by security tools after the execution of an activity. The inputs and outputs of each security tool vary with the activities executed and depend on the runtime environment. Whenever a new security tool is installed in an organization, a security expert can populate the ontological model by defining the capabilities of the installed security tool. More details on the relationship between the classes are shown in Figure 5.4. Figure 5.5 shows some of the instances of the *SecurityTool*, *FunctionalCapability*, *Input* and *Output* class.

5.4.1 Ontological Reasoning

Our proposed *OnSOAR* has a *Reasoner* that uses rule-based reasoning to derive the semantic correlation among the activities, security tool and capabilities. We have defined various rules within the ontology to provide inferred information and some constraints

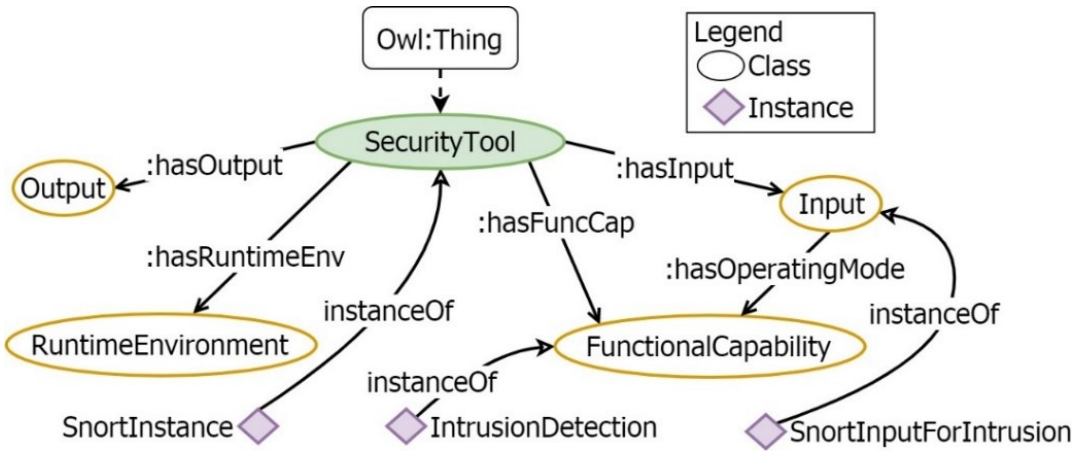


Figure 5.4 The relationship between the *SecurityTool* class and subclass of the *Capability* class

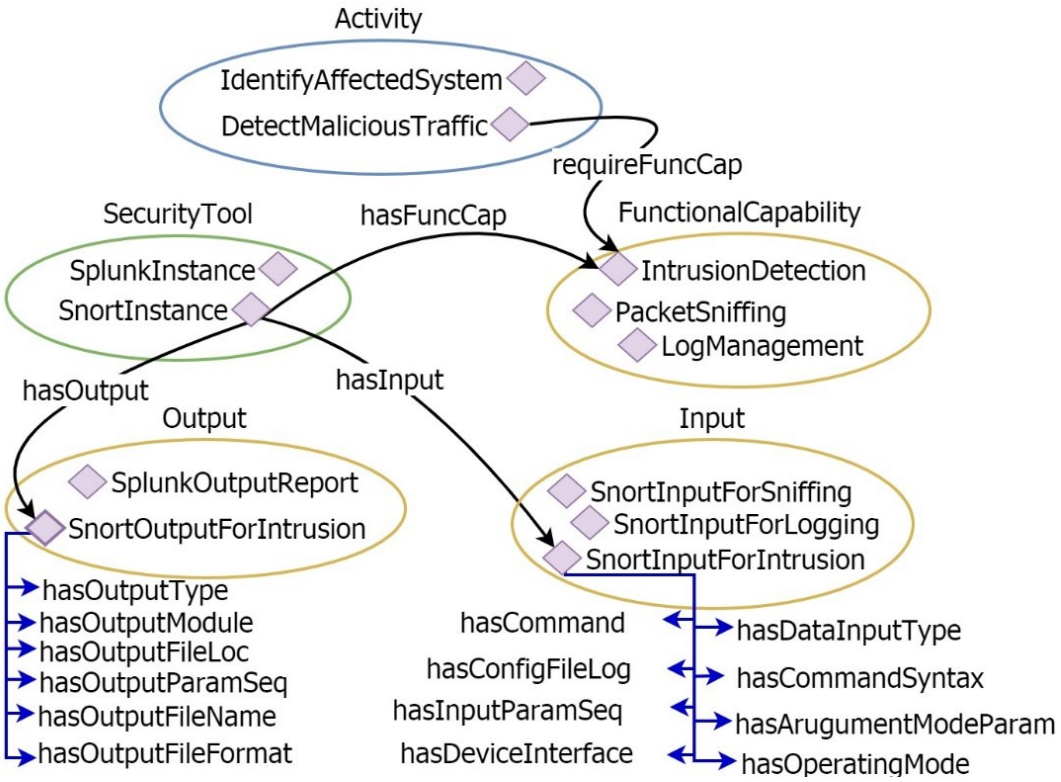


Figure 5.5 Instances of classes of the ontological model and their relationship with other instances. Blue lines represent the data property of the instance of the class Input and Output

for error-free integration. These rules help *OnSOAR* avoid ambiguity, while creating an instance of classes. Based on the rule-based reasoning of an ontology, the *Reasoner* provides the inferred information. For example, Figure 5.5 shows that *SnortInstance*

hasFuncCap *IntrusionDetection*, and hasOutput *OutputForIntrusion*. *OutputForIntrusion* hasOutputType *Alert*. Based on the reasoning, the *Reasoner* infers the relation *SnortInstance* hasOutputType *Alert*. Using the *Reasoner*, we can derive the following information: *"If Snort generates an alert in intrusion detection mode, then the execution of activity DetectIncident by Snort must generate an alert while seeing malicious traffic*".

We provide the examples of some rules (Rules 1 to 9) that are defined in our ontology. The rules enable *OnSOAR* to satisfy the need for reliable automation of the activities. These rules enable us to express the conditions about the occurrence or non-occurrence of auto execution of the activities, the creation of instances, and tracking and managing the activities of IRP. Each security tool must have at least one functional capability to execute an activity (Rule 1). For example, *Snort* must have a functional capability *Intrusion Detection* that can execute an activity *detection intrusion*. Auto-execution of an activity requires at least one system with the functional capability required to execute the activity (Rule 2 & 3).

The input and output of security tools need to be explicitly defined to be integrated into a SOAR. For example, every security tool must have an input command (Rule 4) so that *OnSOAR* can automatically invoke a security tool to execute different types of activities. Most of the output produced by different security tools needs to have an *output file location* from where the SOAR reads the file to interpret the output (Rule 5). For example, if *Snort* runs in *intrusion detection* mode and generates output, then the output type must be an *Alert* (Rule 6).

Rule 1: SecurityTool hasFuncCap min 1 FunctionalCapability
Rule 2: Activity requireUseOf exactly 1 SecurityTool
Rule 3: Activity requireFuncCap some FunctionalCapability
Rule 4: Input hasCommandSyntax some xsd:string
Rule 5: Output hasOutputFileLoc some xsd:string
Rule 6: OutputForIntrusion hasOutputType only Alert

We have also defined the rules for each security tool class. Rules 7 to 9 are examples of such rules, where Rules 7 and 8 are dedicated to security tool SIEM and Rule 9 is for IDS. We impose the criteria to categorize a security tool under a subclass

of the *SecurityTool* class. For example, using Rules 7 and 8, we restrict the creation of a SIEM system instance. Any instance of a SIEM system must satisfy Rules 7 and 8. *OnSOAR* executes the activities sequentially; that causes the input to execute ac_{j+1} which relies on the output of ac_j. As a result, a security tool that is selected to execute ac_{j+1} must have access to the output of a security tool that executes ac_j. For example, if *Splunk* needs to analyze the alert log produced by *Snort*, it must have access to the output file of *Snort*. As per our rules, if the *Splunk* input type is equivalent to the *Snort* output, then the *Splunk* input file location must be the same as the *Snort* output file location. Similarly, *OnSOAR* needs to have the authorization to invoke and stop every security tool that is integrated into its platform.

Rule 7: SIEM hasFuncCap min 3 FunctionalCapability
Rule 8: SIEM hasFuncCap some (EventManagement or EventMonitoring or LogAnalysis or LogCollection or LogManagement)
Rule 9: IDS hasFuncCap only (IntrusionDetection or PacketLogging or PacketSniffing)

5.4.2 Querying the Ontology

The semantic layer deploys a *Query engine* to extract the necessary features from the ontology. The *Query engine* is responsible for communicating with our ontology. It queries the ontology based on the requirement of an *Interpreter*. We designed a set of queries for *OnSOAR* to retrieve the necessary information from the ontology. The queries have three different structures, depending on the required information, as shown in Table 5.1. The *Interpreter* of the orchestration layer invokes the appropriate query to select the security tools (Q_1), functional capabilities required by the activities (Q_2), and capabilities of a security tool (Q_3). Query Q_3 has three different structures: query to extract the input details, query to extract the output details and query about the runtime environment. If an incident response process has an activity *DetectIncident*, and execution of that activity requires the capability *IntrusionDetection*, then the *Query engines* queries the ontology for a security tool that has the capability *IntrusionDetection*. Assuming an instance of *Snort* is available, the query returns *SnortInstance*.

Table 5.1. Different types of query

Query Type	Query Details
Q_1	Query to identify the functional capability required to execute an activity ac_j.
Q_2	Query to search for a security software system s_i that has functional capability $\mathbb{F}$.
Q_3	Query to retrieve the non-functional capability required by s_i to execute functional capability F_a

5.5 SECURITY TOOL LAYER

The security tool layer consists of the various security tools. The security tools send outputs to the *collector* of the data processing layer. The integration layer lies between the security tool layer and the data processing layer, which mainly consist of the integration mechanism through which security tools are integrated (chapter 3). Raw events, system logs, network packets, alerts, security incidents, configuration changes, or commands from experts are sent from the security tool layer. The security tool layer also consists of tools to integrate the knowledge in the ontology and security tool. An ontology *Editor* can also be deployed in the security layer to create, update and modify the ontology classes. In this thesis, we consider a security team will update the ontology's details. Chapter 6 provides a set of APIs that can be used by security teams to interact with the ontology. *OnSOAR* uses the reasoner in every step to check the consistency of the operation that is performed via the editor. For example, if the *editor* attempts to create an instance of *Snort* with functional capability *MonitorFile*, the reasoner considers the ontology inconsistent because, based on rule 9, *Snort* has three capabilities that do not include *MonitorFile*.

5.6 DATA PROCESSING LAYER

This section describes the process of automating the integration of security tools performed in a data processing layer which is coordinated by the orchestration layer in four stages: (i) interpretation of incident, (ii) identification of activities and functional capabilities required to respond to an incident (iii) selection of security tools, and (iv) formulation of command to invoke a security tool. The data processing layer mainly

consist of a collector, output handler and input handler. Given the logs collected from different security tools, the *Collector* pre-processes the raw events data before sending data to the *Output handler*. The output handler annotates the output with the context details, such as *types of log* (i.e., Syslog, server logs, event logs, and message logs), the *location* from where the logs (i.e., the directory) have been collected, the *environment* (OS, endpoint, sensor, server) and timestamps.

5.6.1 Interpretation of the Incident

The *output handler* of data processing receives the output of a security tool from the *Collector*. Upon receiving the *alert* event, it sends the output (i.e., *alert log* produced by s_i *Snort)* to the *Interpreter* to interpret the incident type *I*. Figure 5.6 shows an example process of interpreting the incident type from the alert log generated by a security tool. To semantically interpret the incident, the interpreter first identifies a security tool s_i that generates the alert. It invokes the *Query engine* to get the output format of s_i. Upon receiving the output, it semantically annotates the incident type *I* among the list of features in the alert. The *Interpreter* returns the annotated alert (e.g., alert type, description, and source IP) to the *Output handler* and sends the incident *I* to the *Orchestrator* to take the response action.

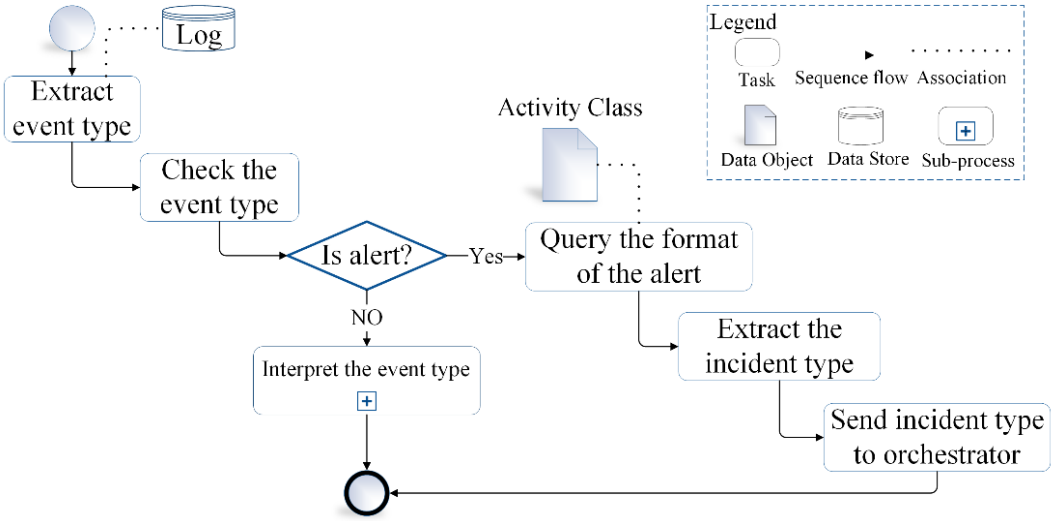


Figure 5.6 Example of sub processes of the integration process for interpreting the incident

5.6.2 Identification of Capability to Respond to an Incident

Upon receiving the incident, I, the orchestrator looks for the possible IRP in the incident response playbook. Assuming $irp_k = \{ac_1, ac_2, ac_3\}$ is the IRP for incident I, the *Orchestrator* extracts the list of activities from irp_k and invokes the *interpreter* to identify the functional capability required to perform an incident response against an incident. For each activity, the *Interprete*r invokes the query engine to run query Q_1 (

Table 5.1), which returns the functional capability required to execute an activity and send it to the *Orchestrator*.

5.6.3 Selection of Security tools

According to the proposed scenario (5.2.2), the auto-execution of the IRP requires *OnSOAR* to identify security tools$_{i,}$ with the functional capability $F_{a,}$ to execute the activity. The query Q_2 is used to identify a security tool with functional capability F_a, where $F_a \in \mathbb{F}$ (Figure 5.7). Each security tool has a relationship with the functional capability class. The *Interpreter* queries the ontology to find the list of security tools required to execute $irp_{k.}$

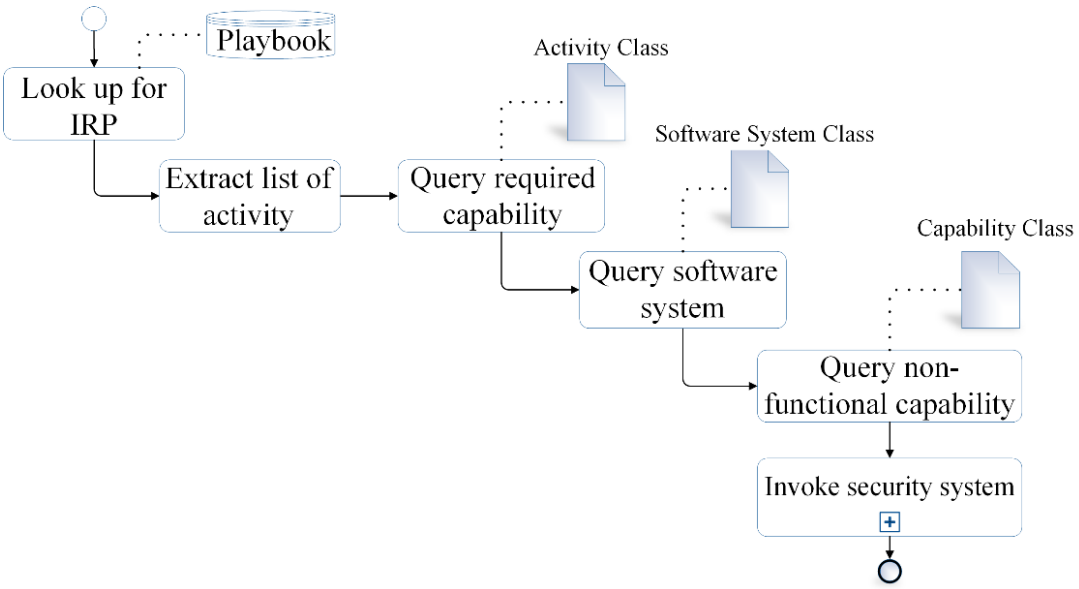


Figure 5.7 Example of sub processes of the integration process for identification of capabilities to automatically respond to an incident

5.6.4 Formulation of Commands to Invoke a Security tool

The *Input constructor* requires the knowledge to formulate the instruction to run s_i in F_a mode. The formulation of commands requires the *Orchestrator* to invoke the interpreter to interpret the input features of a security tool. The ontological model has that information as a data property in the *Input* class (5.4.1.3). The *Interpreter* invokes query Q_3 of *Query engine* to extract the details of the input command. The annotated inputs are passed to the *Input constructor,* which ultimately generates the command details to invoke a particular security tool (Figure 5.7).

In some cases, the execution of specific activities requires the output from previous activities. The *Input constructor* needs to generate the script to invoke a security tool and sends an integration command to execute the activities sequentially. To execute a sequence of activities, the *Input constructor* needs the annotated output of the previous activities. The *Output handler* is invoked to send the annotated output to the *Input constructo*r to execute this activity. *OnSOAR* also has a set of rules to manage interoperability among the security tools. The *Orchestrator* controls the flow of the operation in these cases and invokes appropriate modules, rules and queries to automate the execution of irp_k. The orchestration layer enables direct interpretation of the input and output of security tools in order to make them interoperable.

This whole process of integration of security tools is automated by *OnSOAR* using the proposed ontology, a set of rules and queries.

5.7 EXPERIMENTAL DESIGN AND SETUP

In this section, we describe the experimental setup used for demonstrating and evaluating *OnSOAR*'s ability to automate the process of integrating various security tools. We have developed the required components based on Figure 5.2.

5.7.1 Gathering Input Data for OnSOAR

We gathered a list of activities and identified the functional capabilities required to execute these activities. Table 5.2 shows the part of the activities gathered. The activities were extracted from the IRP for different types of attacks. We extracted the IRP from the website ServiceNow for different incidents. We slightly modified the IRP to match with the capabilities of the security tools we were using. We used the IRP for *DDoS* attack shown in

Table 5.3 for our experiment. The purpose of this experiment is to demonstrate *OnSOAR's* ability to automate the integration process that enables auto-execution of the IRP in a SOAR platform.

Table 5.2. Functional capability mapped with activity

Activity		Functional capability	
ac_1	Detect incident	Intrusion detection	F_1
ac_2	Collect alert log	Log collection	F_2
ac_3	Identify affected system	Alert Analysis	F_3
ac_4	Generate incident report	Report generation	F_4
ac_5	Sniff network packet	Packet sniffing	F5
ac_6	Log network packet	Packet Logging	F_6
ac_7	Isolate affected node	Node Isolation	F_7
ac_8	Kill malicious process	Process killing	F_8
ac_9	Generate report	Report Generation	F_9
ac_{10}	Monitor Event	Event monitoring	F_{10}
ac_{11}	Manage log	Log management	F_{11}
ac_{12}	Investigate alert	Alert analysis	F_3
ac_{13}	Generate alerts	Intrusion detection	F_1
ac_{14}	Scan endpoint	Intrusion detection	F_1
ac_{15}	Remove malware	Process killing	F_8

Table 5.3. Use case scenario with IRP

Incident type		Incident response plan (IRP)
I_1	DDoS attack	$ac_2, ac_{12}, ac_5, ac_3, ac_{13}, ac_{10}, ac_{7,}\ ac_9$
I_2	Malicious process	$ac_{10}, ac_{11}, ac_3, ac_9, ac_8$
I_3	Malware	$ac_{14}, ac_{12}, ac_{15}, ac_9$

5.7.2 Application Environment Setup

To set up the application environment, we chose three different types of security tools, *Snort* as IDS, *Splunk* as SIEM and *Limacharlie* as EDR, that have the capabilities to

execute the activities of Table 5.2. Among these security tools, *Snort* and *Limacharlie* are open source security tools, whereas *Splunk* is commercial. We used the free trial of *Splunk* enterprise version due to its wide range of functional capabilities. We mapped the functional capability of the security tools with Table 5.2, which gave $\delta_{\text{Splunk}} = \{F_2, F_3, F_4, F_9, F_{10}, F_{11}\}$, $\delta_{\text{Snort}} = \{F_1, F_5, F_6\}$ and $\delta_{\text{Limacharlie}} = \{F_2, F_7, F_8, F_{12}\}$. Both *Limacharlie* and *Splunk* can perform other activities that are not listed here. We used a centralized directory to collect alerts logs produced by *Snort*, the event and process log sent by a *Limacharlie* sensor to a Limacharlie cloud and gathered the reports generated by *Splunk*. We defined variables to preserve the information following from the application layer to orchestration layer, for example, variable *SystemFrom* to store security tools that produced the output and variable *filePath* to store the location of the output file (Sec 5.5). We installed the *Snort* and *Limacharlie* sensor applications on the local host. *Limacharlie's* cloud server and *Splunk's* server application were deployed on a virtual machine. We also defined the detection and response rules for *Limacharlie* and *Splunk*.

5.7.3 Development of the Ontological Model

We developed the ontology of *OnSOAR* using protégé [184], an ontology editor. We defined the details of the abovementioned three security tools, the capabilities of these systems and activities of the incident response plan in the ontology. We used RDF/XML serialization to store the ontology. We followed the similar approaches discussed in section 5.4.1. We used the OntOlogy Pitfall Scanner [185] to evaluate the functional and structural dimensions, conciseness, and completeness of the ontology. We maintained the consistency of the ontology, while developing the concepts and populating it. We defined the reasoning rules discussed in section 5.4.1 and used Pellet reasoner to remove any ambiguity. Whenever any class instance does not satisfy the rules imposed on the class, the ontology becomes inconsistent. The reasoner generated error notifications if it found any inconsistency within the classes of the ontology. A violation of the restriction imposed on the classes also caused the reasoner to give an error notification. Thus, the reasoner ensured the consistency of our developed ontological model. We developed an *Interpreter* to semantically interpret the input and output and communicate with an orchestrator. We also developed a *Query engine* that can access the ontology through

SPARQL queries and an interpreter. We loaded the ontology meta model into an Apache Jena Fuseki server, an open source SPARQL server.

5.7.4 Development of the Data Processing Layer

Language. We have used both Java and Python for developing *OnSOAR*. *Splunk* has an SDK that allows interfacing with Java-based programs. It has a REST API to send a command to a security tool through HTTP requests. *Limacharlie* has a Python API available on GitHub. It is simply an interface to a REST API service. Whilst the API provides an easy way to interface with *Limacharlie* services, it is not tool agnostic. Because the *Limacharlie* API has been implemented in Python with the *Limacharlie* package, the *Limacharlie* class in Java is not able to send commands to endpoints by itself. It needs to execute Python scripts and passes the required arguments to the appropriate command to an agent on an endpoint.

Module. We defined a task associated with each activity, as discussed in section 5.2.1. We developed rules for the *Collector*, *Output handler* and the *Input constructor* to collect the events log, interpret output and issue commands to invoke security tool(s). We developed the processes described in sections 5.6.1, 5.6.2, 5.6.2, and 5.6.4 for automating the process of integration. For example, to interpret the output of security tools, the developed *interpreter* extracted the information about a security tool's capabilities and returned hash maps, which contain the information about the output file's location and the action that generated the output.

5.7.5 Baseline Approaches

We used two baseline approaches to perform comparative analysis: a manual integration process (BL1) and an API-based integration process with a static interpreter (BL2).

In BL1, the response process depends on a human. The security team performed the tasks using the security tool. For example, during the monitoring process, if the security team found alerts, they looked in the playbook for the response actions or used previous experience to investigate the alerts.

In the BL2 solution, we developed a set of APIs between security tools to automate the sequence of activities. This process needed pre-developed APIs in both directions for each security tool, i.e., API to send data from *Splunk* to *Limacharlie*, from *Snort* to *Splunk*

and *Limacharlie* to *Splunk* as well. The goal of these APIs was to capture the essential expected capabilities of each security tool that allowed the implementation of the interface by any that kinds of security tools. We developed a static interpreter along with an output handler to automate responses for the *DDoS* attack in

Table 5.3.

5.8 EVALUATION

We have evaluated *OnSOAR* using the following Research Questions (RQ).

5.8.1 RQ1: How Effective is OnSOAR's Process for Automating the Integration of Security tools?

Motivation. *OnSOAR* leverages the semantic interpretation of both activities and security tools' capabilities to integrate security tools through the orchestration process. However, the integration process still works if the APIs are designed and developed between security tools. Thus, we would like to investigate whether the combination of semantic integration and orchestration results in a better-automated process. The RQ1 answers how effective *OnSOAR* is in making security tools interoperable where one system can directly use the output of another system as its input. It also investigates whether or not the system interprets the output of a security tool to formulate the input of another security tool with different capabilities.

Approaches. We used network traffic with malicious behavior that has DDoS attacks. The Snort security tool generated alerts for the security incident I_1 *DDoS* attack, which triggers the whole integration process. We compared the process for *OnSOAR* with the two baseline approaches. We monitored the actions performed by human experts for each activity for the IRP of I_1. We also considered the different numbers of APIs needed for automation of the process using the BL2. Finally, we investigated the developed *OnSOAR* to execute the same list of activities.

Results. Considering all three approaches use the same IRP, for each alert the security team first searched for the alert types and then looked for the possible list of actions and, based on the list, performed the actions. For the standard case, the staff used their previous experience to select security tools to perform each activity. For example, to perform the activities ac_2 and ac_{12} , which are *collect* and *investigate alerts logs,* the

security team collected the alerts generated by *Snort*, and then uploaded those alerts to *Splunk*. For this, the experts needed to log into *Splunk* manually and upload the alert logs and then define the rules to investigate alerts. In case the same types of alerts were seen next, the security team needed to go through the same manual process again. The expert also needed to read the reports generated by *Splunk* and then send the commands to *Limacharlie* to isolate the affected nodes. For similar types of alerts, the manual process requires staff to repeat the same sequence of actions, which requires huge amounts of man-hours and also delays the response process.

For BL2, APIs were available to perform the same sequence of actions. The APIs used a shared directory, where Snort stored the alerts and *Splunk* collected those alerts. A separate collector needed to be designed for each security tool to automate the process and interpret which security tool produced the alerts. BL2 also required definitions of the APIs for each function before the execution of an IRP. For example, execution of I_1 required the design of eight different APIs for each activity, even though the number of security tools was three. If the same set of operations needed to be performed in a different host and server, the APIs would need to be redesigned to work with that host and server.

With *OnSOAR*, once *Snort* generated the alerts, the *Interpreter* automatically identified the incident as a *DDoS* attack and triggered the incident response process. We discuss the effectiveness of *OnSOAR* regarding the challenges mentioned above. *OnSOAR* chose the security tool based on its functional capabilities. It gathered the list of the security tools that can perform the activities in the IRP for incident I_1. The *input command syntax* has the command needed by the security tools to run the security software. The *commandSyntax* has the sequence of the parameters to formulate the commands. Retrieving this information, *OnSOAR* successfully generated the scripts to run *Snort*, *Splunk* and *Limacharlie* in a different mode. Thus, *OnSOAR* successfully invoked each security tool to perform the sequence of activities without the intervention of human experts. Our developed system executed all the activities where it generated three different types of command to run *Splunk* in three different modes. We state that the program removes the operational silos by enabling the security tool to execute different activities.

Given the developed *OnSOAR* selected Splunk to execute the activity ac_2, it generated the configuration commands for *Splunk* to collect the *Snort* log. It queried the *Ontology* to identify the *input command* to invoke *Splunk* to gather *Snort* alert logs, as well as the location of the output of the *Snort* alert log. Based on the results, *OnSOAR* generated the scripts to run *Splunk* to collect *Snort* output. Finally, it invoked Splunk to analyze logs (ac_3) that were based on the rules defined in *Splunk* to identify the affected assets or cluster malicious assets. *OnSOAR* interpreted the output of the security tool to formulate the input of another security tool with different capabilities. A similar process is performed when *OnSOAR* selects *Splunk* to identify the malicious nodes and *Limacharlie* to isolate those nodes (ac_3). *OnSOAR* extracted the node details from *Splunk* and generated the commands details for *Limacharlie*. Through the same process, the system auto-executed a sequence of activities where the output of one system has been used by another system. The system is able to interpret the data generated by the security tool and also interpret the actions performed by the security tool. With BL_2, the same API that was used to send the *Snort* log to *Splunk* was not applicable. To interpret and extract the message of *Splunk,* BL_2 needed to define the rules in the interpreter and then develop the API to send a message to *Limacharlie.*

From the evaluation of *OnSOAR* in comparison with traditional approaches, we state our proposed approach for automating the process of integrating security tools successfully executes the IRP. The same process can be used to automate a different number of IRPs with different types of security tools, whereas, with BL2, new rules and APIs needs to be defined for different IRPs.

5.8.2 RQ2: How Efficient is OnSOAR for Practical Use?

Motivation. During the ontology development process, *OnSOAR* needs a domain expert to design the classes of security tool capabilities. Developing the ontology requires a substantial understanding of the security tools being used. Another time-consuming process is designing the incident response plan and orchestration process. If *OnSOAR* cannot alleviate challenges related to the manual integration process, and substantial efforts are needed to build the ontology, security experts may not be willing to use it in practice.

Approaches. We introduced new incidents (I_2 and I_3) which includes a list of activities and investigates the results of *OnSOAR*. Among these activities, some activities were executed during incident I_1, and some are new. For the new activities, we compared the amount of effort required for *OnSOAR* in terms of the information an expert needed to include in the ontology and the efforts of BL2 in terms of the number of new APIs needing to be designed.

Results. For all the three incidents, the security team required a substantial understanding of the security tools. Also, for both BL2 and *OnSOAR,* the IRP and the automation processes must be defined. For BL2, the security team needed to select security tools, develop the APIs accordingly, and then define the rules to automate the process. For *OnSOAR*, the staff only need to define the capabilities of the security tools to execute those activities. For the already-defined capabilities, no further action needs to be taken. The same integration process of extracting incidents, selecting security tools, interpreting output and formulating input works for executing the IRP for I_2 and I_3. The evaluation shows that little effort and few changes are required in *OnSOAR* to change the IRP.

5.8.3 Threat to Validity

Our work is focused on security tools that are used by the SOC of an organization. Gathering the security tool capabilities was challenging, as the information is not freely accessible. The developed system is limited to security tools that are widely used, freely available and open source. Currently, the evaluation of the proposed method is carried out in a University laboratory environment, which also limits the scope of the experiment. The proposed evaluation approach does not provide any quantitative measurement, which we plan to carry out in future work.

5.9 RELATED WORK

Much research effort is dedicated to using security ontologies to formalize several concepts in the cyber security domain [83, 179-181]. Such concepts include security mechanisms, security objectives, attacks, alerts, threats, vulnerabilities, and countermeasures. Most studies are focused on modelling various types of attacks using ontologies [83, 186], whilst others use ontologies to detect and prevent attacks [180, 187]. None of the abovementioned ontologies can be used by SOAR to make security tools

work together as these ontologies do not have the capabilities of security tools to streamline the incident response process.

Several studies [172, 180, 181] have developed ontologies to formalize heterogenous threat intelligence information for cybersecurity tools. These studies focus on providing effective ways of information sharing and exchanging among cybersecurity communities. One study [172] has developed a Unified Cyber Security (UCO) ontology by combining and mapping widely used ontologies, i.e., STIX and CybOX. UCO provides a standard semantic representation of cybersecurity tools for information integration and cyber situational awareness. Although UCO has enriched the threat vocabulary, it does not provide any support for interoperability among heterogeneous security tools. The work also lacks ontologies that SOARs can use to interpret the activities performed by different security tools to make them work together as a requirement for auto-execution of IRP.

Recently, a set of ontologies have been developed to enable tool-as-service (TSPACE) for a cloud-based platform [158]. Ontologies are used to select and provision tools, based on a stakeholder's requirements, and semantically integrate the artifacts of the tools. A platform provides the stakeholder with a set of tools by using the ontology proposed by the authors; further stakeholders use those tools to perform the required activities. Ontologies help stakeholders to deal with semantic conflicts that arise while integrating multiple tools in the same platform. The ontologies of TSPACE do not provide any support to automate the execution of activities or make the tools interoperable. Thus, the ontologies are not applicable to SOAR. The features of the tools, needed to make them interoperable and remove the operational silos, are not captured in the ontologies of TSPACE. Unlike this generic work, our proposed *OnSOAR* provides the features of security tools that address the issues of interoperability and interpretability, and an automated process for integrating security tools, which can execute their respective activities for working together without human intervention.

We have not found any other work that addresses the interpretability and interoperability issues, which need to be addressed to integrate, automate and orchestrate security tools' activities for seamless incident response processes. Our thesis supports the interoperability and interpretability issues by mapping the capabilities of the security

tools with the activities of the IRP and providing an orchestration process to automate the execution of the IRP. Our ontology formalizes the security tools in detail by capturing their functional and non-functional features. *OnSOAR* can interpret the capability of security tools and generate the commands required to invoke a tool. Hence, it is uniquely positioned to address the challenges. To the best of our knowledge, this is the first work that has developed an ontology to automate the process of integrating security tools in SOAR. The automation is achieved by enabling interpretability, interoperability, and removing operational silos from multivendor heterogeneous security tools.

5.10 CHAPTER SUMMARY

We propose an ontology-driven approach to automating the process of integrating different security tools in a security orchestration platform. By formalizing the concepts of security tools, we aim to support automation in the integration process for security tools that further enables *interoperability* among different security tools. We provide an ontological model that characterizes all the concepts and relationships of SOAR platforms that are required for the integration process. We assert that *OnSOAR* can *interpret* the semantics of the output shared by different security tools and formulate the input required by security tools. Furthermore, *OnSOAR* glues security tools to execute the incident response process automatically. We have demonstrated the viability of the proposed approach by developing and using a proof-of-concept system. The results show that *OnSOAR* can (i) interpret the output of security tools, (ii) invoke a security tool to analyze the data of anothersecurity tool and (iii) automate the integration process to execute an incident response plan. We assert that our approach can minimize the challenges characterized by the manual integration process and effectively automate the integration process of different security tools. The findings from developing and evaluating our approach enable us to believe that *OnSOAR* can be easily integrated with an existing SOAR platform for the large-scale realization of security orchestration and automation in an organization's SOC.

Chapter 6

Declarative API for Security Orchestration Platforms

The proposed semantic-based integration framework (chapter 4) and automated process for integrating security tools (chapter 5) has simplified the way security tools are combined in a security orchestration platform to automate an Incident Response Process (IRP). As discussed in the previous chapters, the emerging and dynamic threat landscape changes the underlying execution environment (i.e., security tools, IRPs and components) of a security orchestration platform. However, the users of security orchestration solutions have difficulty in adapting to these changes because of the ad-hoc and complex architecture of such platforms. This chapter introduces a **Dec**larative API-driven **Or**chestration approach, **DecOr**, to resolve these difficulties. DecOr forms the abstraction layer of our proposed layers in chapter 3. DecOr comprises of (i) three sets of declarative APIs to encapsulate the activities related to security orchestration, (ii) a semantic framework to support an Artificial Intelligence (AI)-enabled design and generation of declarative APIs from task descriptions, leveraging Natural Language Processing (NLP) techniques, and (iii) a semi-automated approach to identify the concepts of an ontology from the available playbooks (i.e., an automated IRP) that are required by a security orchestration platform to automatically interpret the generated declarative APIs. We experimentally evaluate the effectiveness and efficiency of our proposed approach based on a benchmark of 147 task and declarative API pairs that are curated from a set of real-world playbooks. The evaluation results show that DecOr accurately generates declarative APIs in near real-time, with precision and recall values over 80%.

6.1 INTRODUCTION

The dynamic and unpredictable threat landscape causes constant change in the Incident Response Process (IRP) and security tools, requiring the security orchestration and automation platform (also known as Security Orchestration, Automation and Response (SOAR or SOA)) to be easily adaptable and modifiable to the needs of a Security Operation Centre (SOC) [21-23, 33, 159, 188].

The process of updating the relevant components of a SOAR platform is time-consuming and may hinder faster responses as it involves interactions between different levels of expertise and continuous human interventions [22, 26, 189]. In addition, a SOAR platform is difficult to maintain, as it requires an extensive understanding of the underlying libraries and infrastructures to accommodate changes. We assert that the changes associated with the existing SOARs are usually designed in an ad-hoc manner by blending several software components (i.e., proprietary, open-source or third party) through a complex architecture. We believe that, despite the inherent complexity of the existing solutions it should be possible to hide the complex design of SOARs behind an easy-to-use and flexible user interface, so a security team does not need to worry about the details of the libraries, plugins and tools to be used for an IRP.

The architectural complexity of a SOAR platform comes from different factors. For example, to automate the execution of IRPs (i.e., defined in natural language), a SOAR's or playbook's developers build the automated workflow (i.e., playbook) or write executable scripts for IRPs [21-23]. The developed automated workflow is mostly static and cannot be changed with changing needs. Besides this, multiple security tools are required to execute an IRP. Security teams are expected to modify playbooks when security tools are installed and/or modified. It is quite difficult to seamlessly modify and develop new IRPs and integrate new security tools without fully knowing the playbooks or libraries of a SOAR [22, 26, 189]. A security team needs to have extensive domain expertise and understanding of the available tools, libraries and security requirements for developing IRPs. A security team in a SOC uses SOAR for different purposes, such as network administration or incident response planning. However, it should not be necessary for him/her to have extensive domain knowledge about the underlying execution environments.

Another factor contributing to the complexity is enabling interpretability and interoperability between security tools and the SOAR. Semantic technologies (e.g., ontological knowledge bases, RDF and OWL) are usually leveraged to support interpretability and interoperability among heterogeneous security tools and components [21, 23, 159, 188]. In chapters 4 and 5, an ontology has been designed to semantically describe the concepts of security tools, their actions, relationships between different concepts and the categories of the concepts. Whilst the existing semantic-based solutions automate the interpretation of security tools through semantic integration [21, 23, 159, 188], human experts still need to update the ontology whenever changes occur. Development of an ontology requires substantial understanding of the domain of security orchestration. Furthermore, to integrate an updated IRP, a human expert needs to map that IRP with suitable concepts from an ontology, which further requires them to identify the semantic concepts and the relationship between the different categories of the ontology.

In this chapter, we propose an Artificial Intelligence (AI) enabled **Dec**larative API driven **Or**chestration approach, namely **DecOr**, to design a flexible, scalable and easy to modify SOAR to overcome the above-mentioned challenges. DecOr enables end-users of a SOAR to focus on the WHAT (e.g., run a playbook or block an IP) through a set of **declarative APIs (dAPI)** that hides a SOAR's operating complexities at different levels of abstraction. The use of AI techniques, such as Natural Language Processing (NLP) and semantic technologies, in designing dAPI enables easy adoption of changing IRPs and security tools, which also contributes towards automating the manual and labor-intensive tasks. The dAPIs are designed to provide the security team with *flexibility* to define IRPs or execute their desired tasks without having detailed knowledge about the underlying libraries and tools. We identify three sets of dAPIs: (i) *orchestration API*, (ii) *integration API* and (iii) *execution API,* as shown in Figure 6.3 (discussed in section 6.3.2).

DecOr's input can be a text-based query or commands to perform a task in response to a security incident. An organization's security team can provide a description of the tasks or use a dAPI to update IRPs or provide commands. We have categorized the main operations of DecOr into two phases: understanding the commands or tasks and

interpreting dAPIs. For the first stage, we developed a semantic framework, SecAPIGen, that accepts a command or task description and translates that into a dAPI. SecAPIGen uses NLP techniques to interpret a command or task of SOAR and automatically generates a dAPI. For example, to invoke a task "*block the IP address using the Checkpoint Firewall*", the generated dAPI is *"block.IP.firewall(Checkpoint)".*

For the second stage of our DecOr, we have developed a semi-automated approach, SemOnto, to identify concepts from IRP playbooks for the relevant ontology, which are required to interpret a dAPI. SemOnto automatically extracts the properties of the generated dAPI elements, which are incorporated in the form of the concepts of an ontology. In this way, SemOnto reduces the burden on security teams, who manually analyze and identify concepts for an ontology. We have modified the existing ontology of a SOAR to map the generated dAPI elements with the classes of an ontology. Mapping dAPIs with an ontology allows automated interpretation of dAPIs by a SOAR. Hence, AI-enabled declarative API-driven orchestration, DecOr, enables automated adaption and integration of IRPs and security tools by hiding the underlying complexity of the SOAR.

We have evaluated the effectiveness and efficiency of our approach using 147 tasks from 194 playbooks. Our evaluation explores three key questions: (i) *How effective is SecAPIGen in generating and identifying dAPIs for different tasks?* (ii) *Can SemOnto identify the concepts of an ontology from a playbook?* and (iii) *How efficient is DecOr in terms of required time?* The results of our evaluation demonstrate the effectiveness of using AI-based approaches (i.e., NLP and semantic technologies) to automate the generation of a dAPI from a task description in terms of accuracy and response time. Across the benchmark, DecOr generates dAPIs with precision and recall above 90% and 80% respectively. On average, for 90% of the cases, DecOr successfully identifies the properties of the generated dAPI elements as a concept of ontology. Moreover, the average response time to generate different parts of a dAPI is close to 170 milliseconds. The following are the key contributions of this work:

- A set of requirements for designing three sets of dAPIs for AI-enabled dAPI driven orchestration to integrate security tools and execute IRPs in a SOAR.

- A semantic framework, SecAPIGen, to automate the generation of dAPIs from task descriptions by leveraging NLP tools and techniques. SecAPIGen allows both technical and non-technical users to interact with a SOAR without requiring detailed knowledge about the underlying libraries and configurations.
- A semi-automated approach, SemOnto, to identify the properties of the generated dAPI elements as a part of the concepts of an ontology from the playbooks of a SOAR.
- Design and execution of rigorous evaluation of the developed solution using 194 playbooks.

In section 6.2, we provide the background and preliminaries of security orchestration and ontologies, and a detailed motivation scenario for proposing declarative API driven orchestration. We describe the proposed approach in Section 6.3. The experiment design, proof of concept system and evaluation of our proposed approach are discussed in Section 4. Section 5 discusses the benefits, limitations, and future directions and opportunities. Section 6.6 provides the related works. Section 6.7 concludes the chapter.

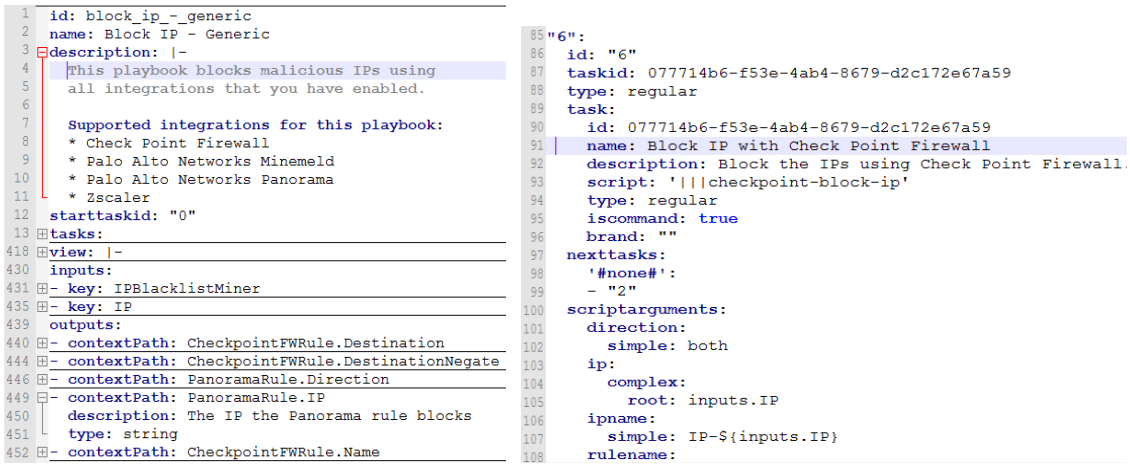
6.2 PRELIMINARIES AND MOTIVATION

This section provides the background information on the playbooks and ontology of a SOAR platform. We start with an example playbook and provide a brief overview of the ontology proposed in chapters 4 and 5. Then we introduce a running example that we use to illustrate the motivations of this chapter. Finally, we formulate the problem and introduce the key notations that are used throughout this chapter.

6.2.1 Playbook for Security Orchestration and Automation

A SOAR platform executes playbooks to respond to specific incidents. The playbooks contain automated workflows designed from an IRP of an incident. The SOAR platform developers or playbook designers explicitly code all possible action flows and forecast all possible exceptions in a playbook. Figure 6.1 shows a code snippet of a playbook from Demisto (i.e., a SOAR platform) [166]. The playbook script is written in YML [190] and expresses the code for "*blocking malicious IP using all available security tools*". Figure 6.1(a) shows that a playbook has a set of tasks (line 13), inputs (line 430)

and outputs (line 439). Figure 6.1(b) shows a snippet of one of the tasks (line 91) of that playbook. It shows that, to execute a task, a SOAR platform runs a script (line 93). The details of the script, such as names and arguments, are also defined in a playbook (lines 100-108). The playbook also contains the names of the security tools (lines 8-11) that will be used to execute a task (Figure 6.1(a)). For example, for a task *block IP* (Figure 6.1(b)), the security tool is "*Checkpoint Firewall*". Several things can go wrong while executing a playbook, such as a task may fail due to the unavailability of the scripts or argument settings of security tools; or a security tool might not have the authority to block a particular IP, making it impossible to execute a task and so forth.



(a) An example of the script of a playbook – *Block IP* (b) Code snippet of task "*block the IPs using Check Point Firewall*"

Figure 6.1 Example (a) snippet of a playbook for block IP which contains the list of tasks, inputs and outputs of a playbook and (b) snippet of a task of a playbook to run a script to "block IP with Check Point Firewall", where the task consists of the script arguments that are required to execute it

6.2.2 Ontological Knowledge Base

Whilst a playbook helps to bring automation in a SOAR platform, the interpretability and interoperability of a SOAR platform are achieved by formalizing and storing the semantic knowledge of security tools and IRPs in an ontological knowledge base. A SOAR platform can automatically use a knowledge base to interpret the data generated and ingested by heterogeneous security tools. We have analyzed different security tools and built an ontology based on different security tools' and playbooks' activities (refer to chapters 4 and 5). The key concepts in an ontological knowledge base are security tools, their capabilities and tasks of IRPs (see Figure 6.2(a)).

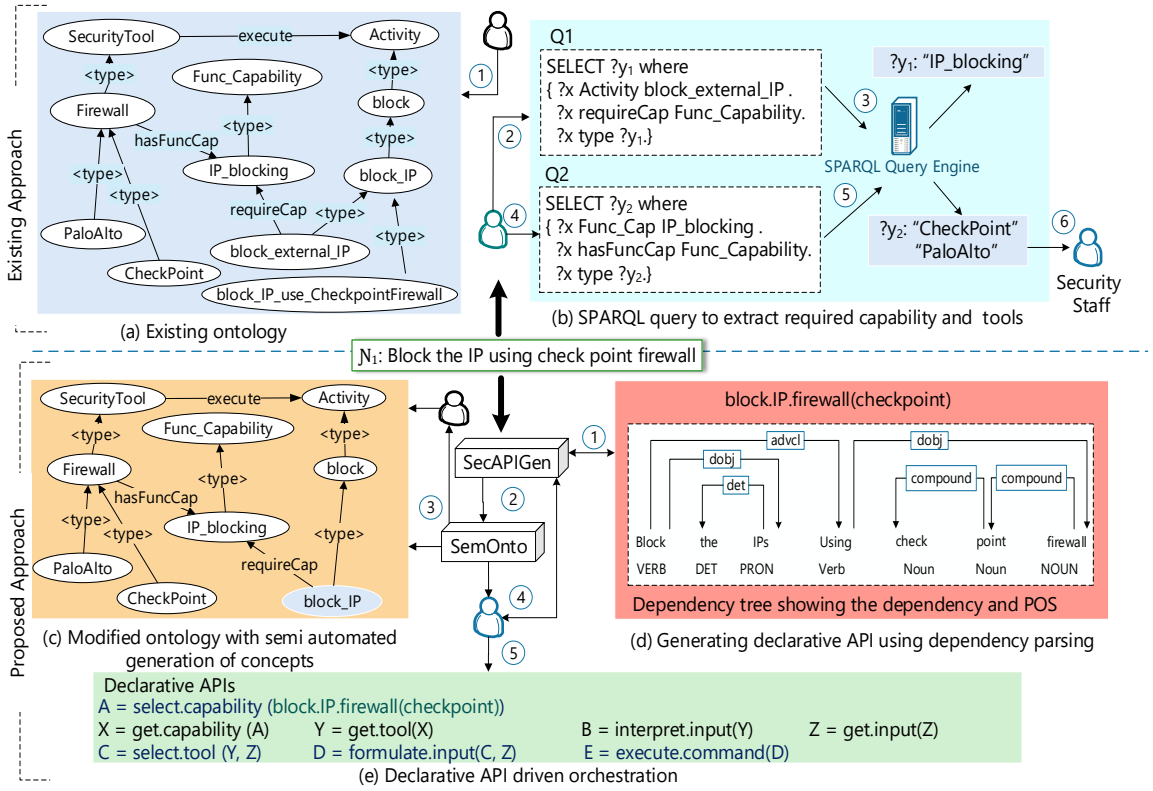


Figure 6.2 An example execution of a command based on the existing approaches of semantic interpretation and integration and our proposed approach

Figure 6.2(a) shows an excerpt of an ontology for a SOAR platform proposed in chapter 4. The ontology of Figure 6.2(a) has three key classes: *SecurityTool*, *Activity* and *Func_Capability*. The classes are designed in such a way that a SOAR platform can easily select the required category of security tools and automatically interpret their capabilities (i.e., functionality, inputs and outputs) to execute a task. Security tools such as *Checkpoint* and *PaloAlto* are categorized as a *Firewall* under the *SecurityTool* class (see Figure 6.2(a)). The *Func_Capability* class presents the capabilities of security tools that are required for the execution of an activity. The *Activity* class defines the tasks of a SOAR platform (i.e., defined in a playbook) and actions of security tools. The task descriptions are mapped with the *Activity* classes of an ontology. For example, the key task of Figure 6.1 (i.e., *blocking malicious IP using all available security tools)* is mapped with *block_IP*, which is a subclass of *block* that is categorized under *Activity* class.

An ontology also shows the relationships (presented using an edge) between classes. For example, Figure 6.2(a) shows the relationship between the *SecurityTool* class

and *Activity* class is *execute.* The *SecurityTool* class also has a relationship, *hasFuncCap,* with *FuncCapability* class. The ontology of Figure 6.2(a) further shows that the execution of an activity *block_IP* needs the capability *IP_blocking* and security tools of type *Firewall* that have that capability. Considering these relationships between different classes and the properties of each class, a SOAR platform interprets that *Checkpoint* and *PaloAlto* (i.e., a type of Firewall) can execute the activity *block_IP*. Furthermore, the rules are defined to maintain the consistency of the ontology. Using an ontology, a SOAR platform generates the scripts to invoke security tools by querying inputs of security tools that are selected to execute a task. To enable interpretability and interoperability, an ontology needs to be continuously updated as a result of changes in IRPs and security tools. A SOAR platform may fail to interpret the available data or may select the wrong security tools if its ontology is not up to date.

The following subsection presents the difficulties with managing and updating existing playbooks and ontology-based approaches adopted by a SOAR platform with a scenario. Using the same scenario, it also highlights the potential benefits of using a set of dAPIs for orchestration in a SOC.

6.2.3 Motivation Scenario

We provide a running example to illustrate the limitations of the existing approaches (i.e., playbooks and ontological knowledgebases) that are used to automate the execution of IRPs and enable semantic interpretations and integrations of security tools in SOAR platforms. We also show the motivation for, need and importance of the reported work on dAPI.

Example 1. An organization needs to defend against DDoS (i.e., Distributed Denial of Service) attacks. The idea is to design an IRP to scan the organization's host or endpoints for malicious activities, quarantine the affected endpoint and, if required, block the malicious IPs. We assume an IRP is available to periodically scan an organization's infrastructure for malicious activities and isolate or quarantine the affected endpoints. We further consider that such a system would require an IRP, a network to be scanned and a security tool (or tools) to be used. The requirements can be gathered and implemented in multiple ways to achieve the overall goal, which can be accomplished in several ways. There might be a preferred ordering among different tasks that builds

orchestration (e.g., the task of visualizing the alerts data comes after the task of scanning). In particular, some workflows are alternatives, while others operate in a particular sequence.

We assume an IRP provided by an incident response planning team is (i) *scan an organization network*, (ii) *analyze the traffic*, (iii) *identify the affected endpoint*, (iv) *isolate endpoint* and (v) *block malicious IPs*. A playbook developer designs a playbook for the corresponding IRP as an orchestration of the existing security tools' capabilities and events. They may design different playbooks for each step, where the execution of each step requires the following of a sequence of tasks. For example, Figure 6.1 shows a playbook for a task (v) *block Malicious IPs* of an IRP. A SOAR platform or security team executes the playbook when required. However, the emergent threat behaviors cause a continuous change in a SOAR platform's IRPs. As shown in Figure 6.1, a playbook is a hardcoded script or code that is designed to execute a sequence of tasks. Security teams may want to execute part of a playbook during the analysis of an incident, for example, if security teams just want to perform the task shown in Figure 6.1(b). For this, a separate playbook is needed that will perform the task in Figure 6.1(b). This incurs an overhead of defining all the possible combinations of tasks in a playbook or designing a playbook at runtime. Knowing all possible sequences of tasks is not possible due to the emergent threat behavior. On the other hand, defining new playbooks requires knowledge about the underlying libraries and security tools' APIs.

Another alternative approach is to semantically represent and store security tools' capabilities and tasks from playbooks in an ontology (as discussed in section 6.2.2) instead of designing all possible playbooks. For this approach, we assume an ontology designer formalizes the available security tools, their capabilities and tasks of the related IRPs using semantic knowledge (e.g., Figure 6.2(a)). The corresponding query is also designed to extract the relevant classes and instances from an ontology. Figure 6.2(b) shows examples of SPARQL queries that are used to query an ontology. In many instances, security teams need to select from the alternatives which tools to use, and modify queries to get the required features. A SOAR platform is unable to execute IRPs or interpret the security tools' capabilities if these are not explicitly defined in an ontology. For this reason, an ontology needs to be updated with changes in any playbook

or IRP. Adding a new IRP, modifying an IRP, or adding/modifying a security tool will require the addition of a definition in an ontology if the tasks or security tools are new. Irrespective of the approaches used (i.e., playbook or ontology), security teams should be able to use a SOAR platform to modify and update IRPs and add new security tools. Furthermore, a SOAR platform should be able to interpret an IRP and automate its execution.

We assume a security team wants to express a task in the natural language form $\mathcal{N}_1$ = "*Block IP with Check Point Firewall*" instead of the task (v) *block malicious IPs*. An ontological knowledge base of the existing security tools, their capabilities and activities is available and $\mathcal{N}_1$ is mapped with the *Activity* class *owl_ac1 = "block_IP_use_checkpointFirewall"* (step 1 Figure 6.2(a)). Next, the SPARQL queries of Figure 6.2(b) are executed to identify the capability *"IP_blocking"* (step 2) and security tools with that capability (step 4). The classes associated with the security tools (i.e., *Palo Alto* [191] and *Checkpoint* [192]) are suggested to the security team (step 5)**.** The security team then selects a suitable class that represents the requested security tool. A relationship is defined between *checkpoint* and *owl_ac1* to automate the execution of $\mathcal{N}_1$ with *Checkpoint*. If a SPARQL query is not available to identify a specific security tool for a specific capability, the security team needs to design a suitable query, which requires an understanding of the ontology and SPARQL.

We address these issues by designing a framework, SecAPIGen, to generate different elements of a dAPI from $\mathcal{N}_1$. Figure 6.2(d) shows the dependency parsing of $\mathcal{N}_1$. The SecAPIGen extracts the semantic relationships, based on the dependency parse tree of $\mathcal{N}_1$ for generating dAPIs (step 1 Figure 6.2(d)) which is "*block.IP.Firewall (Checkpoint)*". SecAPIGen recommends a set of dAPIs to the security team (step 4), through which they modify the plan, define new plans, integrate security tools and even update the ontology (step 3). We resolved the issue of manual identification of an ontology's classes for each command by developing SemOnto, a semi-automated approach to identify an ontology's concepts. We modified the existing ontology (see Figure 6.2(c)) using SemOnto, which maps the dAPIs with the classes of an ontology (steps 2 and 3) and recommends a potential list of classes and their properties to the security team (Step 3). SemOnto makes subclasses of *block_IP,* shown in Figure 6.2(a),

redundant. Using the dAPIs, the security team can easily execute task $\mathcal{N}_1$ by providing the sequence of commands shown in Figure 6.2(e). The complexity of interacting with the ontology, modifying the IRPs or selecting a security tool to execute a task is hidden through dAPIs. The proposed approach of using dAPI-driven orchestration to execute $\mathcal{N}_1$, as shown in Figure 6.2(e), also hides the complexity of interacting with different components of the SOAR platform and makes the SOAR platform easy to manage and interact with. In the following section, we formulate the components that we have discussed previously (in sections 6.2.1, 6.2.2 and 6.2.3) and which are required to explain our proposed approach, DecOr.

6.2.4 Problem Formulation

We formulate the problem considering the scenario discussed in section 6.2.3, where an organization already has a set of security tools, S= $\{S_1, S_2 \ldots S_i \ldots\}$, a set of playbooks, $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2 \ldots \mathcal{P}_m \ldots\}$, and an ontology O. Examples of S_i include Snort, Splunk, Firewall and so forth. Each playbook, $\mathcal{P}_m$, provides an automated workflow of an IRP with a set of tasks T, where $T = \{T_1, T_2 \ldots T_n \ldots\}$. Each task T has an unstructured text description $\mathcal{N}$, input and output. Each IRP is a set of tasks and their descriptions, where the IRP = $\{\mathcal{N}_1, \mathcal{N}_2 \ldots \mathcal{N}_k \ldots\}$. For simplicity of the experiments, this work only considers a task description that contains an imperative and a simple sentence structure [193, 194]; hence, we have not considered any complex or compound sentences (detailed in section 6.4.1.3). The tasks of a playbook are mapped with the security tools' capabilities. The function $\delta(S_i, T_n)$ defines that a security tool, S_i, has the capability to perform a task T_n. The set of security tools, tasks and capabilities are formalized in an ontology under the class *SecurityTool* (owl_st), *Activity* (owl_ac), and *Capability* (owl_cap) respectively. We consider owl_class to be the set of different types of classes of O, where $owl\_class = \{owl_{st}, owl_{ac}, owl_{cap} \ldots owl\_input, owl\_output, owl\_re \ldots\}$. For simplicity, again, here we consider three major classes: owl_st, owl_ac, and owl_cap. The goal of this work is to provide a dAPI-driven orchestration approach for the end-users of a SOAR platform that enables easy ways to manage, interact and update IRPs and security tools. Given an IRP or a single task description $\mathcal{N}$, we identify the set of dAPIs (detailed in section 6.3.2) that are required to execute $\mathcal{N}$.

Definition (Declarative API-driven Orchestration). In declarative API-driven orchestration, users provide a command (or a task) through a set of dAPIs, without specifying the detailed steps and rules for its execution. The complex details are hidden behind the commands. The security team do not define the sequence of actions that are needed to execute the task.

Table 6.1 shows the summary of notations that are used in this chapter.

Table 6.1 Summary of Notations

Notation	Meaning	Notation	Meaning
SOAR	Security orchestration and automation	$\mathcal{N}$	Task description in natural language
SOC	Security operation center	δ	Capability function
IRP	Incident response process/ plan	T	Set of tasks
AI	Artificial intelligence	$\mathcal{A}^{\mathrm{I}}$	Set of integration APIs
NLP	Natural language processing	$\mathcal{P}$	Set of playbooks
dAPI	Declarative APIs	O	Set of ontologies
Synset	Synonym set	a_i^{D}	An element of $\mathcal{A}^{\mathrm{D}}$
S	Set of security tools	a_{ij}^{D}	j$^{\text{th}}$ part of declarative API a_i^{D}
$\mathcal{A}^{\mathrm{D}}$	Set of declarative APIs	a_k^{ξ}	An element of $\mathcal{A}^{\xi}$
$\mathcal{A}^{\xi}$	Set of execution APIs	a_j^{O}	An element of $\mathcal{A}^{\mathrm{O}}$
$\mathcal{A}^{\mathrm{O}}$	Set of orchestration APIs	a_j^{I}	An element of $\mathcal{A}^{\mathrm{I}}$

6.3 OUR APPROACH

This section presents our AI-enabled **Dec**larative API-driven **Or**chestration approach, **DecOr**. We first provide the overall structure of a system overview of Decor. We then introduce (i) *dAPIs* that are designed to hide the inherent complexity of a SOAR from a SOC (ii) *SecAPIGen*, a framework that automates the generation of the dAPIs from task descriptions and (iii*)* ***SemOnto***, which **Sem**antically interprets the concepts of an **Onto**logy related to dAPIs, and recommend a possible set of an ontology's classes.

6.3.1 Overview

Figure 6.3 provides an overview of our proposed approach for AI-enabled **Dec**larative API-driven **Or**chestration, **DecOr**. It comprises three core components: dAPIs, SecAPIGen, and SemOnto. DecOr is built on top of an existing SOAR. Based on an in-depth analysis of the SOAR's activities, we designed three sets of dAPIs. Our approach

supports interactions between the SOAR and the security team through these three sets of dAPIs.

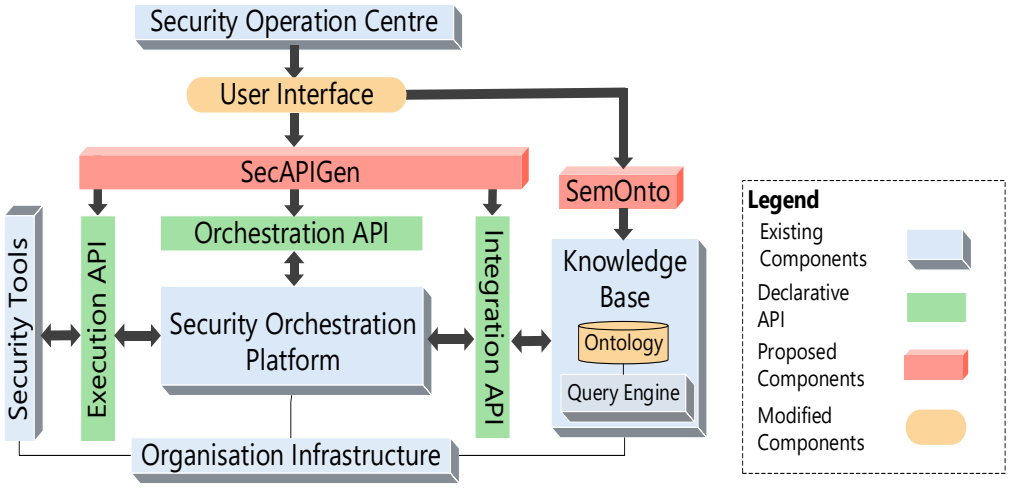


Figure 6.3 System overview of DecOr; security tools, security operation center, the playbooks, knowledge base and organizational infrastructure form the underlying execution environment of a SOAR platform

Declarative APIs (dAPI) (section 6.3.2) The interactions among different components of the SOAR and SOC are maintained using dAPIs. DecOr has three sets of declarative APIs: ***orchestration APIs***, ***integration APIs*** and ***execution APIs***, which are designed to manage tasks related to orchestration, integration and automation, respectively. The set of *orchestration APIs* is mainly defined to support interaction between the security team and the SOAR platform to define or update IRPs or running playbooks. Some of examples of orchestration APIs include "*scan.traffic(malicious).IDS*", "*block.IP*", and "*run.playbook(alert.enrichment)*". We design integration APIs to integrate and update an ontology's concepts. Some examples of the tasks that can be encapsulated through integration APIs include querying, updating, or crafting the concepts of an ontology. The dAPI "*get.tool*" is an example of an integration API. The set of execution APIs is designed to automatically interpret the data generated and ingested by multiple security tools to execute an IRP. The tasks encapsulated through the execution APIs are interpretation of the data generated by security tools, identification of the commands' parameters, formulation of the commands to invoke security tools and so on. The dAPIs "*interpret.input*", and "*formulate.command*" are the examples of execution APIs.

SecAPIGen (section 6.3.3): SecAPIGen is a semantic framework that generates, manages and recommends dAPIs leveraging NLP technologies (i.e., tools and techniques). *SemAPIGen* provides a user interface to the SOC (Figure 6.3) through which the security team can define new plans or request execution of an incident or a playbook. Common forms of user interfaces are GUI, interactive dashboard or an IDE that provides access to a dAPI and renders the activities' responses according to the security requirements. We consider two modes of a user interface: a *novice mode* for non-technical users and an *expert mode* for technical users. On novice mode, SecAPIGen receives text-based commands or imperative sentences (i.e., $\aleph_1$ = *block the IPs using the checkpoint Firewall*) from the SOC. The commands or requests are passed to *SecAPIGen*. Upon receiving $\aleph_1$, SecAPIGen interprets the commands and identifies that a *block* operation needs to be performed by a security tool, *Checkpoint Firewall,* on an *IP*. On expert mode, the security team gives commands directly through the dAPIs (e.g., *block.IP.Firewall(Checkpoint)*) to SecAPIGen. Table 6.2 shows examples of dAPIs and the corresponding task descriptions.

SemOnto (section 6.3.4): SemOnto gives SOC the flexibility to define their ontology or modify the bootstrapping ontology (i.e., the existing ontology) with changing playbooks and IRPs. The interactions of SemOnto and other components are exposed through an integration API. SemOnto uses the integration API to query an ontology to interpret the security tools' data and integrate the knowledge about security tools with the ontology. To execute dAPIs, a SOAR platform needs to understand the input and output of the different components. SemOnto automatically identifies these details and relationships among different components from the playbook of the SOAR platform. For example, to perform a task "*block the IPs using the checkpoint Firewall*", the SOAR platform needs to interpret the IP and its properties and the Firewall's inputs, which SemOnto automatically extracts from the available playbooks.

6.3.2 DecOr Declarative API (dAPI)

We propose a semantic framework, *SecAPIGen*, leveraging NLP technologies to design and generate a set of declarative APIs (dAPIs). We define a dAPI by combining similar functions and mapping from a high level. For example, both the tasks "*block malicious IPs*" and "*block the IP with Checkpoint Firewall*" come under the dAPI "*block*". The

dAPI is used to provide direct commands without highlighting how the commands need to be executed. In this section, we discuss the design and categories of dAPIs.

6.3.2.1 *Description of dAPI*

We consider $\mathcal{A}^{\mathrm{D}}$ as a set of dAPIs. Each dAPI, $a_i^{\mathrm{D}} \in \mathcal{A}^{\mathrm{D}}$ has three parts $a_{i1}^{\mathrm{D}}, a_{i2}^{\mathrm{D}}$, and a_{i3}^{D}, as shown in Table 6.2. The first part, a_{i1}^{D}, consists of the key abstract functions such as *block, scan, verify* and *detonate*. The second part, a_{i2}^{D}, provides an object (e.g., *IP* and *capability*) on which a task needs to be performed. The third part, a_{i3}^{D}, identifies the specific elements corresponding to an object (i.e., *endpoint* and *ontology*), which provide fine-grained details about a task or the types of tools (i.e., *firewall*) to be used. The last two parts of a dAPI take parameters that provide the modifiers or attributes of an object and its components. For example, the task "*block external IPs in the Firewall*" specifies the type of IPs. To capture these types of information, we designed a_{i2}^{D} and a_{i3}^{D} to take parameters. Table 6.2 shows an example, "*block.IP(external).firewall*" where a_{i2}^{D} takes *"external"* as a parameter. We have followed a systematic approach and proposed the semantic framework SecAPIGen to generate a set of $\mathcal{A}^{\mathrm{D}}$ (the details are given in section 6.3.3). Enabling auto-creation of the APIs reduces human efforts to identify libraries manually ident to execute a task. In the following subsections (i.e., sections 6.3.2.2, 6.3.2.3 and 6.3.2.4), we present the three sets of $\mathcal{A}^{\mathrm{D}}$.

Table 6.2 Examples of declarative APIs

	Description	**Declarative API**		
		First Part	**Second Part**	**Third Part**
a_1^{D}	Block the external IPs in the Firewall	block	IP(external)	firewall
a_2^{D}	Interpret the capability of checkpoint Firewall	interpret	capability	firewall(checkpoint)
a_3^{D}	Check the activity class block in the ontology	check	activity(block)	ontology

6.3.2.2 *Orchestration API to Update and Define New Plans*

A SOC's security team (e.g., an IRP planner) uses a set of orchestration API $\mathcal{A}^{\mathrm{O}}$, where $\mathcal{A}^{\mathrm{O}} \subset \mathcal{A}^{\mathrm{D}}$ and $\mathcal{A}^{\mathrm{O}} = \{a_1^{\mathrm{O}}, a_2^{\mathrm{O}} \ldots a_i^{\mathrm{O}} \ldots\}$ to execute, modify or update IRPs without having a detailed knowledge of the available security tools. Using $\mathcal{A}^{\mathrm{O}}$, a security team provides

commands to the SOAR platform or designs IRPs. $\mathcal{A}^{O}$ is built based on the available IRPs and task descriptions of the existing playbooks. Each orchestration API a_i^{O} is dedicated to perform a specific type of task, such as *scan, block* and *retrieve*, as shown in Table 6.3. Each API element has several variations, depending on the objects on which a task is performed or the security tool that is used to perform a task. For example, Table 6.3 shows two variations of an API element *block*, where a_1^{O}: *block.IP.firewall(checkpoint)* and a_2^{O}: *block.IP(external).firewall*. In both instances, three elements of the two dAPIs are similar; they vary in terms of their parameters. $\mathcal{A}^{O}$ is used to execute or update the existing plans or define the new plans of the SOAR platform. The elements of dAPIs are mapped with *classes* of an ontology. For example, the first part of the dAPI is categorized and mapped with *Activity* class, owl_ac. In this way, an ontology is modified and designed to have three levels of class, such as a class *block* that has subclass *block_IP,* as shown in Figure 6.2(c).

Table 6.3 Examples of the selected set of orchestration APIs

API	Example API	Description
block	block.IP.firewall(checkpoint) block.IP(external).Firewall	Perform block operations on IPs using different types of security tools.
scan	scan.networktraffic scan.Endpoint	Scan network traffic, host and other resources for malicious behavior.
retrieve	retrieve.information.account(user)	Retrieve information from different data source(s).
verify	verify.account(source.user).address (email)	Verify information such as account details, email address, credentials and so forth.

Unlike the existing ontology, which has a separate class for *block_IP_firewall*, the modified ontology omits these sets of classes by mapping the activity with the corresponding security tools. The details of a *firewall* are mapped under the *SecurityTool* class, which enables us to systematically define the classes of an ontology and remove ambiguity. In most cases, security teams use the orchestration API to interact with a SOAR platform to execute or define an IRP or run a playbook. Changing threat behavior, integration of new tools or modification of IRP, requires frequent access to and updates

of the orchestration API. Thus, changes and updates of $\mathcal{A}^{\mathrm{O}}$ are more frequent than integration APIs and execution APIs.

6.3.2.3 *Integration APIs to Communicate with an Ontology*

The goal of integration APIs is to automatically incorporate security tools' details into and from an ontology. We design a set of integration APIs $\mathcal{A}^{\mathrm{I}}$, where $\mathcal{A}^{\mathrm{I}} \subset \mathcal{A}^{\mathrm{D}}$ and $\mathcal{A}^{\mathrm{I}} = \{a_1^I, a_2^I \dots a_i^I \dots\}$ to interact with an ontology and integrate the security tools' data. An existing ontology is modified and extended to incorporate auto-identification of semantics from the available playbooks of the SOAR platform. The interaction of the SOAR platform with the ontological knowledge base is encapsulated through $\mathcal{A}^{\mathrm{I}}$, which encapsulates certain tasks such as *querying an ontology*, *updating an ontology* or *crafting the concept of ontology by analyzing a playbook*. $\mathcal{A}^{\mathrm{I}}$ enables a SOAR platform to integrate, interpret and select a security tool based on an IRP's commands, and frees the security team from knowing the details of the underlying query language (e.g., SPARQL) or ontological knowledge base. For example, DecOr queries an ontology using integration API "*get.securityTool*". Table 6.4 shows examples of $\mathcal{A}^{\mathrm{I}}$. For instance, if a SOAR platform needs a security tool with the capability *block*, an ontology has a class *block*. We have designed an a_i^I "*check*" to find out whether such a class exists in an ontology and then another a_i^I "*get"* to query an ontology and retrieve the security tools' details. An integration API "*update*" is used to update the features of a particular class of an ontology, such as modify an ontology or add a new feature. Unlike "*update*" or "*post*", dAPI is used to add new classes in an ontology.

6.3.2.4 *Execution API to Invoke Security Tools*

Execution APIs play an important role in making the process of integration and execution seamless and automated. We have leveraged the process of automating the integration of security tools proposed in chapter 5 and designed a set of execution APIs $\mathcal{A}^{\xi}$, where $\mathcal{A}^{\xi} \subset \mathcal{A}^{\mathrm{D}}$ and $\mathcal{A}^{\xi} = \{a_1^{\xi}, a_2^{\xi} \dots a_i^{\xi} \dots\}$ to encapsulate the individual tasks of a process. In this way, components can be designed individually without affecting each other's performance or exposing their functionalities through $\mathcal{A}^{\xi}$. For example, a process of automated integration of security tools involves (i) *selecting the security tools*, (ii) *interpreting the security tools' input and output features*, (iii) *checking the security tools'*

capability, and then (iv) *form the input commands to invoke the security tool* and finally (v) *execute a given task*.

Table 6.4 Examples of a selected set of integration APIs

API	Example API	Description
identify	identify.activity.playbook	Identify the activity details from the playbook's description
validate	validate.activity(scan).ontology	Validate the new activity scan in an ontology
update	update.output.ontology	Integrate the newly identified output into the existing ontology
get	get.activity(block)	Query to select a class of an ontology
post	post.input(IP)	Query to create a class in an ontology
delete	delete.tool(snort)	Query to select and remove a class in an ontology
check	check.activity(block)	Check whether a class exists in an ontology

As an example, where one security tool needs to use the outputs of other security tools, the process changes to (i) *interpreting the security tools' outputs*, (ii) *deconstructing the outputs*, then (iii) *extracting the required features,* (iv) *interpreting the features*, and finally (v) *formulating the inputs*. We design $\mathcal{A}^{\xi}$ to hide the details of the process of automating the integration of security tools using semantic knowledge. Some of the examples of integration APIs are "*select.securityTool*" and "*interpret.capability*". The interpretation of the security tools' input and formation of the commands is done through $\mathcal{A}^{\xi}$. Thus, $\mathcal{A}^{\xi}$ enables security teams to control and modify the process based on their required task. Table 6.5 shows examples of $\mathcal{A}^{\xi}$ that we have designed.

Similar to an orchestration API and integration API, an execution API a_i^{ξ} has different parts. The first part a_{i1}^{ξ} defines the main tasks, such as *select* and *interpret*, where the second part a_{i12}^{ξ} defines an object (i.e., *security tools* and *input*) on which a task needs to be performed. We have designed the second part based on the classes of an ontology. As most of the features are extracted from an ontology, an easy to map a_i^{ξ} with an ontology helps to keep task execution seamless and interpretable. An example of a process that uses $\mathcal{A}^{\xi}$ to invoke security tools involves a combination of "*select. tool*", *"interpret. capability"* and *"formulate. input"*.

Table 6.5 Examples of a selected set of execution APIs

API	Example API	Description
select	select.tool.capability select.capability	Select security tools from a list of available tools or tools explicitly specified by the security team, and the capabilities of the security tool required to perform an action
interpret	interpret.output interpret.input	Semantically interpret data generated and ingested by security tools and different components of DecOr
deconstruct	deconstruct.input deconstruct.command	Deconstruct generated outputs or commands to extract required features
formulate	formulate.input formulate.command	Formulate input commands of security tools based on the script's arguments
execute	execute.command execute.script	Send execution commands or invoke security tools by calling appropriate API with passing right arguments in right formats

To select security tools, the first task is to get the list of the available security tools and their inputs, outputs and the runtime environments to check the compatibility of the security tool with the ongoing execution environment. For this reason, a query to the ontology "*get.securityTool.capability(IP_blocking)*" is performed, which returns *Palo_Alto* and *Check_Point*. To interpret security tools' capabilities, an appropriate security tool is selected. Next, to formulate the input, the features of a host are checked, and an input is constructed. The security tools' details are available in the ontology. After formulating the command, "*execute.command*" is used to invoke a security tool that executes the requested task. We leverage the existing work on building a semantic knowledge base (i.e., an ontological model) to unify the heterogeneous security tools in a structured way [171, 172, 188]. Our ontological model stores information about the inputs required to invoke a security tool, for example, function calls, the number of parameters, the list of parameters and also different variations of a single function call.

6.3.3 SecAPIGen: Semantic Framework for dAPI Generation

Figure 6.4 provides the system level overview of the SecAPIGen framework, which generates and recommends dAPIs based on users' requests. One of the main goals of SecAPIGen is auto-generation of dAPIs, along with the parameters to execute a command $\mathcal{N}_{k.}$ There are two aspects of automating the generation of a dAPI. First,

SecAPIGen needs to automate the generation of different elements of a dAPI. Second, based on the generated dAPI, SecAPIGen should be able to identify semantically similar dAPI elements (detailed in section 0) from the available dAPI list. For example, two dAPIs *"quarantine.endpoint"* and *"isolate.endpoint"* are semantically similar and can be mapped as a single dAPI.

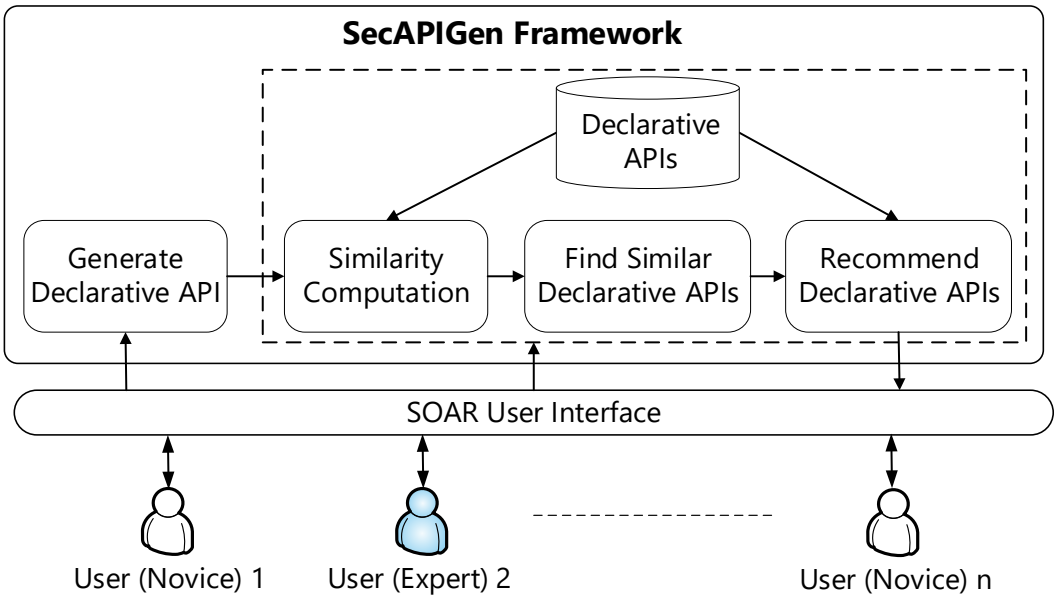
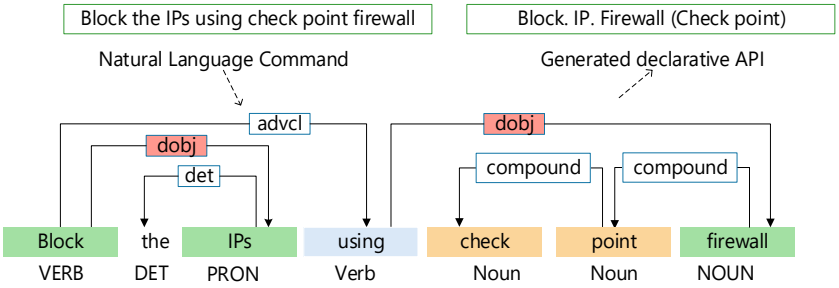


Figure 6.4 System overview of the SecAPIGen framework

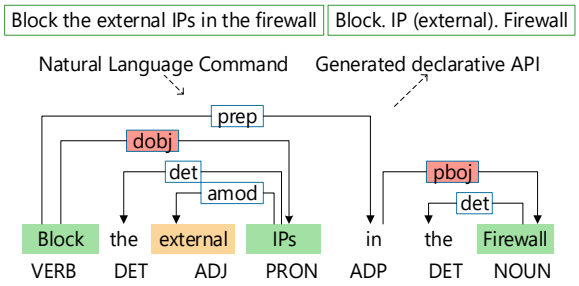
As shown in Figure 6.4, SecAPIGen provides support for both novice (i.e., user 1) and expert users (i.e., user 2). In novice mode, a security team with little or no prior knowledge about dAPIs and security tools directly provide text descriptions of a task (e.g., "block a malicious IP") that they want to carry out. SecAPIGen automatically generates a dAPI from a task description using Algorithm 1 and Algorithm 2. In expert mode, a security team who frequently interact with SOAR call on a dAPI related to a task. For example, for "block the malicious IP", instead of providing a text description, the security team provides the dAPI with *"block.malicious.IP"*. SecAPIGen executes *"block.malicious.IP"* which requires a language model to search for similar dAPIs and recommend a suitable dAPI. Algorithm 3 identifies semantically similar dAPIs and recommends the most similar dAPIs to security the experts. In both modes, modules under the dashed box in Figure 6.4 are executed, where the *dAPI generation* module is only executed for inputs from novice users. In the following sections, we present three algorithms of SecAPIGen, using an example.

6.3.3.1 *Automatic Generation of dAPIs from Task Description*

We use dependency parsing [195, 196] to achieve our goal of automating the generation of different parts of a dAPI, a_i^{D}. Dependency parsing extracts a dependency parse tree of a sentence that represents grammatical structures of a sentence and defines the relationships between a root word of a sentence and the words which modify the root [195, 196]. It identifies both syntactical and semantic parsing of a sentence structure. Syntactical parsing provides a parse tree, whereas the semantic analysis provides the subject, object and different attributes of a sentence.



(a) Dependency tree for "block the IPs using check point firewall"



(b) Dependency tree showing the dependency and POS for "block the external IPs in the firewall"

Token	Token dep	POS of Token	Token head	Token head dep	POS of Token head	Token child
Block	root	VERB	Block	root	VERB	IP, In
The	det	DET	IPs	dobj	PROP	-
External	amod	ADJ	IPs	dobj	PROPN	-
IPs	dobj	PROP	Block	root	VERB	The, External
In	Prep	ADP	Block	root	VERB	Firewall
The	det	DET	Firewall	pboj	NOUN	-
Firewall	pobj	NOUN	In	prep	ADP	the

(c) The different linguistic associated with each word (here token) of sentence "block the external IPs in the firewall"

Figure 6.5 Example of (a) dependency parsing for "block the IPs using checkpoint Firewall" (b) dependency parsing for "block the external IP in the Firewall" and (c) other linguistic features of a token (each word is considered as a token) such as token head, token head dependency, and token child

Given a task description $\mathcal{N}_k$, our problem is to automate the generation of a dAPI. Figure 6.5(a) and Figure 6.5(b) show examples of dependency parsing for two tasks with description $\mathcal{N}_1$ = "*block the IPs using checkpoint Firewall*" and $\mathcal{N}_2$ = "*Block the external IP in the Firewall*". An arrow shows the dependency between two words. Several types of dependency (i.e., nominal subject and direct object) exist between different words of

a sentence [195-197]. We have used the *spaCy* NLP library to identify the dependencies and parts of speech (POS) of different words [197]. The *spaCy* NLP library takes a sentence and returns a dependency parse tree. Each node of a tree corresponds to a word of a sentence and contains several features such as dependency, POS, and the lemmatization form of a word (i.e. lemma). Figure 6.5(c) shows other linguistic features associated with each word (i.e., each node of a parse tree).

The root of a dependency parse tree is considered as an ancestor of all other nodes of a parse tree. From Figure 6.5(a) and Figure 6.5(b), we can see that the root of both sentences is *"block"*. Usually, a root word is used to identify the main verb of a sentence. We consider a root that POS is *verb* as the first part of a dAPI. Among many dependencies, the most common ones are the nominal subject (nsubj) and direct object (dobj). A nominal subject modifies a nonverbal predicate of a sentence, where an object is either a direct object (dobj) of a root or a prepositional object (pobj) [196]. In Figure 6.5, "*IP*" is the dobj of the root. Other characteristics that we have considered are modifiers of roots, objects and subjects, which are defined as a compound dependency or attribute dependency. Another common dependency that we have considered is the clausal complement (ccom) of a word. The compound words are a modifier of a compound word sequence, where an attribute is any miscellaneous properties of an object or subject. Leveraging these properties, we have designed Algorithm 1 to identify three parts (i.e., API elements) of a dAPI, $a_i^{\mathfrak{D}}$ where $a_i^{\mathfrak{D}} = a_{i1}^{\mathfrak{D}} \,.\, a_{i2}^{\mathfrak{D}} \,.\, a_{i3}^{\mathfrak{D}}$.

Algorithm 1 generates a declarative API, $a_i^{\mathfrak{D}}$, utilizing the dependency parsing techniques that give a dependency parse tree of a sentence $\mathcal{N}_k$. It considers the root of the sentence $\mathcal{N}_k$ as the first part ($a_{i1}^{\mathfrak{D}}$) of an API $a_i^{\mathfrak{D}}$ (line 1). From lines 2-25, Algorithm 1 takes each child token of a root and considers the dependencies of each child with root. If a root has a dependency of a nominal subject with its child, only then does Algorithm 1 consider the nominal subject in $a_{i1}^{\mathfrak{D}}$ and the root in $a_{i2}^{\mathfrak{D}}$ (lines 3-5). If a root has a direct object dependency with a child, the word of that child is considered in the second part $a_{i2}^{\mathfrak{D}}$ (lines 6-7). For cases where a child has a dependency of clausal complement with a root, Algorithm 1 takes each node (that is the child of the root's child) of that child separately to identify the possible elements of a dAPI (lines 8-6). For each node of a child with a dependency clausal complement, if a child is a nominal subject, it is considered

as $a_{i2}^{\mathfrak{D}}$ and if a child is a direct object, it is assigned to $a_{i3}^{\mathfrak{D}}$ (lines 11-15). Lines 17-24 consider the children of a child of root that has a dependency of preposition object and assign it to $a_{i3}^{\mathfrak{D}}$. The operation discussed above is repeated for each child token of a root which generates the elements (i.e., $a_{i1}^{\mathfrak{D}}$, $a_{i2}^{\mathfrak{D}}$ and $a_{i3}^{\mathfrak{D}}$) of a dAPI $a_i^{\mathfrak{D}}$.

Algorithm 1. Generating a declarative API from task description using dependency parsing

Input: Dependency tree of $\mathcal{N}_k$ with dependency and Parts of Speech (POS) of each word
Output: Different parts of a dAPI $a_i^{\mathfrak{D}}$
1 **Initialize** $token \leftarrow$ *root* of the dependency tree, $a_{i1}^{\mathfrak{D}} \leftarrow$ extract_text(token)
2 **For each** *child* ∈ *token . children* **do**
3 **IF** *dependency* of a *child* is a nominal subject of *root* and *POS* of that *child* is "verb" or "noun"
4 $a_{i1}^{\mathfrak{D}} \leftarrow$ extract_text(child)
5 $a_{i2}^{\mathfrak{D}} \leftarrow$ extract_text (token)
6 **ELSE IF** *dependency* of a *child* is a direct object of *root* and POS of that *child* is "verb" or "noun"
7 $a_{i2}^{\mathfrak{D}} \leftarrow$ extract_text(child)
8 **ELSE IF** *dependency* of a *child* is an adverbial or clausal complement or modifier of *root* & POS of that *child* is a "verb"
9 Consider the children of *child*
10 **For each** *node* ∈ *child . children* **do**
11 **IF** *dependency* of a *node* is a nominal subject of *child* and *POS* of that *node* is "verb"
12 $a_{i2}^{\mathfrak{D}} \leftarrow$ extact_text(node)
13 **ELSE IF** *dependency* of a *node* is a direct object of *child*
14 $a_{i3}^{\mathfrak{D}} \leftarrow$ extract_text(node*)*
15 **End IF**
16 **End For**
17 **ELSE IF** *dependency* of a *child* is preposition and *POS* of a *node* is "adverbial position"
18 Consider the children of *child*
19 **For each** *node* ∈ *child . children* **do**
20 **IF** *dependency* of a *node* is prepositional object of *child* and *POS* of node is a "verb" or "noun"
21 $a_{i3}^{\mathfrak{D}} \leftarrow$ extract_text(node)
22 **End IF**
23 **End For**
24 **End IF**
25 **End For**
26 **Return** $a_{i1,}^{\mathfrak{D}}\ a_{i2}^{\mathfrak{D}}, a_{i3}^{\mathfrak{D}}$

Each object and subject have further modifiers (i.e., adverbial modifiers, noun compound modifiers), which we utilized to get the properties of an object and subject $\mathcal{N}_k$. Consider an example task "*block the external IPs in the Firewall*", where *external* is a modifier (i.e., adjective modifier: amod) of *IP* (Figure 6.5(b)). Leveraging the modifier

dependencies, Algorithm 2 is designed to identify the parameters of $a_{i2}^{\mathfrak{D}}$ and $a_{i3}^{\mathfrak{D}}$. Lines 1-6 take each of them separately and look for their modifier dependencies from the corresponding parse tree. Several cases exist where a single object or subject has multiple dependencies; for example, "*send a message to the source user email address*". Here, the *"source user email"* is considered as a modifier of the object *address*. Lines 8-21 identify such forms of multiple modifiers of a dAPI element. Algorithm 2 returns the list of parameters for $a_{i2}^{\mathfrak{D}}$ and $a_{i3}^{\mathfrak{D}}$.

Algorithm 2. Identify declarative API (dAPI) parameters using dependency parsing

Input: Generated dAPI $a_i^{\mathfrak{D}} = a_{i1}^{\mathfrak{D}}, a_{i2}^{\mathfrak{D}}, a_{i3}^{\mathfrak{D}}$ for $\mathcal{N}_k$, and dependency tree of $\mathcal{N}_k$ with dependency and Parts of Speech (POS) of each word
Output: List of dAPI elements' parameters

1 **For the** second and third part of $a_i^{\mathfrak{D}}$
2 $token \leftarrow extract\_token(a_{ij}^{\mathfrak{D}})$
3 $token\_param_j \leftarrow$ *identify_api_param* (*token*, $a_{ij}^{\mathfrak{D}}$)
4 remove $a_{ij}^{\mathfrak{D}}$ from $token\_param_j$
5 **End For**
6 Return $token\_param$ list
7 ***identify_api_param*(***token*, *token_parameter*)**
8 Consider the children of *token*
9 **For each** *child* ∈ $token$. *children*
10 **IF** *dependency* of a *child* is noun compound, adjective or adverbial modifier
11 $token\_text \leftarrow$ *extract_text* ($token$)
12 $new\_param\_list \leftarrow$ *extract_text (child)* + . + *token_text*
13 $token\_parameter \leftarrow$ replace $token\_value$ with new_param_list *in* $token\_parameter$
14 Consider the *children* of *child*
15 **For each** *node* ∈ *child. Children*
16 **IF** *dependency* of a *node* is noun compound, adjective or adverbial modifier
17 *token_parameter* = $identify\_api\_param$(*child*, $token\_parameter$)
18 **End IF**
19 **End For**
20 **End IF**
21 **End For**
22 **Return** $token\_parameter$

6.3.3.2 *Identifying Semantically Similar dAPIs*

The problem in this section is to identify whether (i) the *dAPI elements generated using* Algorithm 1 or (ii) a *dAPI element provided by the security team* exists in the

existing set of dAPIs $\mathcal{A}^{\mathrm{D}}$ with a different name. The dAPI elements generated using Algorithm 1 may exist with a different name in the initial set of $\mathcal{A}^{\mathrm{D}}$. The reason behind this is the word ambiguity in defining a task. For example, both $\mathcal{N}_3$: "*quarantine the endpoint*" and $\mathcal{N}_4$: "*Isolate the affected endpoint*", are referring to a similar task. Algorithm 1, generates two dAPI "*quarantine.endpoint*" and "*isolate.endpoint*" respectively for $\mathcal{N}_3$ and $\mathcal{N}_4$. We consider "*quarantine*" and "*isolate*" as semantically similar because both of them ultimately do similar types of tasks. Thus, we propose Algorithm 3 to minimize the number of dAPIs in $\mathcal{A}^{\mathrm{D}}$ by linking these types of semantically similar dAPI elements under a single dAPI element. For example, instead of having both *quarantine* and *isolate*, $\mathcal{A}^{\mathrm{D}}$ has one dAPI element *"quarantine"* in its first part, which is used to perform both $\mathcal{N}_3$ and $\mathcal{N}_4$.

Algorithm 3. Identify semantically similar declarative API (dAPI)

Input: List of an initial set of declarative APIs $\mathcal{A}^{\mathrm{D}}$ and generated declarative API $an^{\mathrm{D}} = an_1^{\mathrm{D}}.an_2^{\mathrm{D}}.an_3^{\mathrm{D}}$
Output: Suggested declarative API as^{D}
Initialize $as^{\mathrm{D}} \leftarrow an^{\mathrm{D}}$, $max\_score \leftarrow 0$, sim_score
IF an_i^{D} not in $\mathcal{A}^{\mathrm{D}}$:
  For each $an_j^{\mathrm{D}} \in a_i^{\mathrm{D}}$:
    For each $a_{ij}^{\mathrm{D}} \in \mathcal{A}_j^{\mathrm{D}}$:
      Syn_api = return the synonym set of a_{ij}^{D} from Wordnet synsets
      Syn_newapi= return synonym set of an_{ij}^{D} from Wordnet synsets
      For each *s_api* ∈ *Syn_api*:
        For each s_newapi ∈ Syn_api:
          $similarity\_score$ = the similarity between s_api and s_newapi
          IF $similarity\_score \geq sim\_score$ and Pos of s_api is equal to Pos of s_newapi:
            IF $similarity\_score > max\_score$:
              $max\_score \leftarrow similarity\_score$
              $as_j^{\mathrm{D}} \leftarrow a_{ij}^{\mathrm{D}}$
            End IF
          End IF
        End For
      End For
    End For
  End IF
Return as^{D}

Algorithm 3 identifies semantically similar words that remove word ambiguity and represent a set of tasks with a minimal set of dAPIs. The reason behind representing an

abstract set of functions with a set of minimum dAPI elements is to keep the ontology and commands (i.e., dAPI) straightforward, which enables security teams to easily accommodate the set of commands supported by DecOr.

Two critical issues with a dAPI generated using Algorithm 1 or provided by expert users are (i) how to identify whether a dAPI element is semantically similar to the available list of dAPI elements and (ii) how to address the ambiguity of natural language $\mathcal{N}$. Algorithm 3 considers each part ($a_{i1}^{\mathcal{D}}, a_{i2}^{\mathcal{D}}$ and $a_{i3,}^{\mathcal{D}}$) of a dAPI $a_i^{\mathcal{D}}$ that is generated using Algorithm 1 (lines 1-2). Algorithm 3 utilizes the concepts of Word-Net, which is a lexical database (i.e., dictionary for the English language) [198-200], mainly used for NLP related tasks. For each part $a_{ij}^{\mathcal{D}}$ of $a_i^{\mathcal{D}}$, Algorithm 3 finds semantically similar words of $a_{ij}^{\mathcal{D}}$ by considering the lexical structure of the two words and the semantics of sense of these words (lines 3-19). For example, NLTK has an interface (i.e., synset) to look up words in WordNet, which returns the instances of a synonymous set (synset) [198-200]. Synonymous words that express similar concepts are grouped together and return as instances of a synset. A single word can have one synset or multiple synsets, depending on the datasets used to build Wordnet.

Algorithm 3 calls *Wordnet.synset(words)* to get a list of synonym words and calculates the similarity score between two words (lines 7-17). The similarity score indicates how close the two words are in terms of their semantics. We consider different types of similarity metrics (e.g., Wu-Palmer metric and Resnik) [200, 201] to compare the semantic similarity (line 9). We define a threshold for the similarity score, sim_score and discard words with a similarity score lower than the threshold (line 10). The similarity function and similarity scores' threshold were chosen during the experiment, as discussed in section 6.4.2.1. The input of Algorithm 3 can also be a dAPI from an expert user. The same set of steps are carried on to identify whether the provided dAPI exists or not. We consider that for some queries or commands a SOAR platform might not have any dAPI. This can happen when a SOAR platform does not have any IRPs or playbooks to perform the requested tasks, or the security tools that are required to execute the requested task.

SecAPIGen automatically generates different parts of a dAPI, leveraging NLP techniques. These API elements are further required to incorporate into an existing ontology [21, 23, 159, 188] of the SOAR platform to keep its ontology up to date and enable automatic interpretation of activities, inputs and outputs. We consider a bootstrapping ontology (i.e., the existing ontology) is available for semantic interpretation of security tools and activities which are utilized by the SOAR platform to execute an IRP [21, 23, 159, 188]. In the next section, we provide a semi-automated approach to identify properties of generated dAPI elements from the detailed descriptions (i.e., inputs, outputs, and scripts) of available playbooks.

6.3.4 SemOnto: Identification of Ontological Concepts from Playbooks

We map the generated dAPI elements with the classes of an ontology, so that a SOAR can interpret a dAPI. For example, as shown in Figure 6.2(c) dAPI "*block.IP*" is mapped with an *Activity* class owl_ac_i: *block* which has a subclass *block_IP*. To select a security-suitable tool for executing "*block.IP*", a SOAR must semantically interpret what an "*IP*" is and find the properties of an IP. The reason is that, instead of "*block.IP*", the command could be "*block.account*". Here, a SOAR needs to semantically differentiate "*IP*" and "*account*", as it might require two different security tools to execute these two tasks. Furthermore, the properties of an *account* (i.e., type, domain, and user) may vary, depending on the task, which might also require different types of security tools to block it. As a result, the ontology of a SOAR needs to have classes that formalize *IP* and *account*. It enables the SOAR to extract information for interpreting and selecting security tools for providing *IP* and *account* as inputs. It also reflects the properties of a particular class of an ontology. For example, consider a class *Account* that has specific properties, such as *type*, *email* and *username*. A playbook has the detail of the input and output descriptions (Figure 6.1), which can be utilized to identify those properties of a dAPI element that are required by a SOAR platform to invoke a security tool.

We provide a semi-automated approach, *SemOnto*, to identify the properties of a dAPI element from available playbooks, which are further incorporated as concepts (i.e., class, subclass and link between two classes) of an ontology. These properties are required by a SOAR platform to automate the interpretation of a dAPI and generate the

commands or scripts required to invoke a security tool for execution of a task. SemOnto analyzes a playbook to extract the key features of a task.

As the structure of playbooks may vary from vendor to vendor, the key features (i.e., input, output and tasks) of a playbook that SemOnto is required to analyze are provided manually by human experts. Each playbook has a sequence of tasks that are performed by different security tools and a SOAR platform. As a result, identifying the inputs and outputs of a task also reflects the inputs and outputs of the security tools. In this work, we have considered the playbooks of Demisto, a collaborative playbook that provides output in the form of an output context path (lines 440-452 of Figure 6.1(a)). The *context path* is a dot-notation representation of the path to access the context object of an incident. SemOnto analyzes the *output context path* of Demisto's available playbooks to understand the key properties associated with a dAPI element (e.g., *IP*, *email* and *account*). We have designed Algorithm 4, which is at the core of SemOnto, to identify the key concepts (i.e., the properties of dAPI elements) of an ontology, relationships between classes and the data properties of a class.

Algorithm 4. Identification of ontological concepts from a playbooks' description

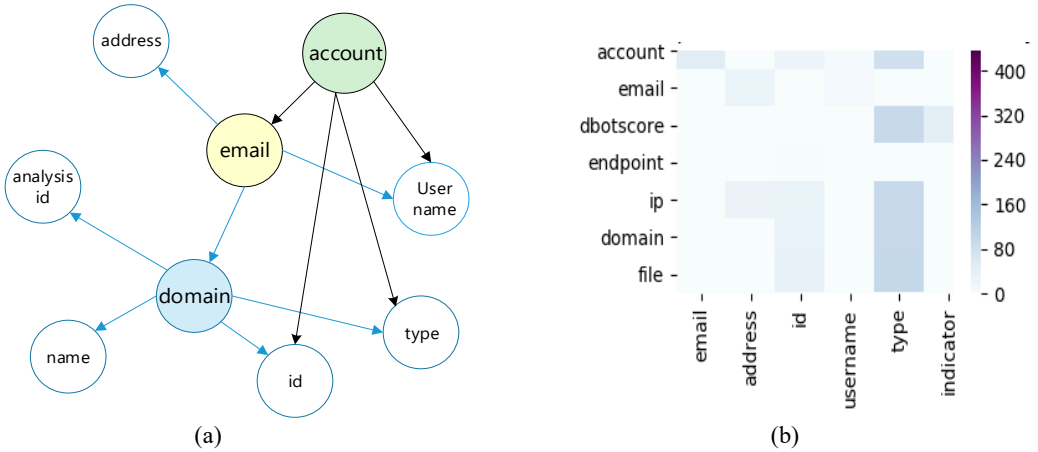
Input: Set of playbooks
Output: Set of classes and class properties
Extract the input, output and context path from the playbook and pre-process the data.
Extract context path from the output and build a dictionary, $List\_obj$ separating the objects in the context path
For each pair of adjacent objects (obj_i . obj_k) in the context path
    Calculate the occurrence of each pair and store in an adjacent matrix Adj_mat.
End For
For each object obj_i in $List\_obj$, consider all the rows and columns in the Adj_mat associated with obj_i
    IF all the row of Adj_mat for obj_i is zero, assign the *type*(obj_i) as a class object
    ELSE IF all the column of Adj_mat for obj_i is zero, assign the *type*(obj_i) as attribute
End For
For each object pair (obj_i obj_k) in Adj_mat where Adj_mat (obj_i obj_k) is not zero*:*
    IF both *type*(obj_i) and *type*(obj_k) are class then obj_i has a relation with obj_k:
        IF the properties of obj_k properties of obj_i, Adj_mat (obj_k obj_i) is 0 then obj_k is a subclass of obj_i
    ELSE IF *type*(obj_i) is a class and *type*(obj_k) is an attribute, then obj_k is a data property of obj_i
End For
Return the class list with the properties

Algorithm 4 takes each context path and separate objects from a context path based on the dot (.) and builds a vocabulary for objects (lines 1-2). We have found that the objects are similar to different parts of a dAPI element, except the first part refers to an action (i.e., block, scan and isolate). Algorithm 4 develops an adjacent matrix Adj_mat that stores neighbors of each object in an ordered form to highlight the occurrence of pairs of objects (lines 3-5). For an ordered pair $obj_i.obj_k$, $Adj\_mat[obj_i][obj_k]$ = total number of occurrences of $obj_i.obj_k$. For example, if a pair "account. email" is seen in 20 output context paths, $Adj\_mat[account][email] = 20$. If "email. account" is not seen in any of the context paths other than Adj_mat[email][account] = 0. The order of the objects' pair in the context path is considered as identifying the classes, their data properties and object properties. The order of the pairs of objects can also be shown as a directed graph (Figure 6.6(a)).

Lines 6-9 consider each object and its adjacency matrix to identify the classes and properties of the classes of an ontology. If all rows of an object are zero, Algorithm 4 considers that object as a class (line 7). On the other hand, if all columns for an object are zero, then that object is considered as a data property (line 8). For example, based on the graph in Figure 6.6(a), which shows the ordered pairs of objects, *account*, *email* and *domain* are considered as classes of an ontology. For each pair, Algorithm 4 uses the adjacency matrix to identify related classes and attributes (both data property and object property) of each class (lines 10-14). If for two objects obj_i and $obj_k, Adj\_mat[obj_i][obj_k]$ is not zero, then it considers a relation (i.e., an object property) exists between these two classes (line 11). If $Adj\_mat[obj_i][obj_k]$ is zero, then obj_k is considered as a data property of obj_i (line 13). Figure 6.6(a) shows an example where *email* is an object property of *account* and *domain* is an object property of *email*. It also shows the *account* class has the data properties *id*, *username* and *type*.

The heat map of Figure 6.6(b) shows the frequencies of each pair of objects that occur together. The x-axis indicates an object and the y-axis indicates the neighbors of that object. It shows which two objects are related. SemOnto generates such heatmaps to provide insight about the related objects that are identified by Algorithm 4. The relationships of Figure 6.6(a) can also be noticed in the heatmap of Figure 6.6(b). SemOnto automatically generates the discussed properties by analyzing playbook data.

It can identify whether two classes are related. However, the name of the object property is not automatically generated by SemOnto, which requires an expert to define the object properties. The heatmap can be used by domain experts to verify the identified concepts and define the relationship of the related classes.



(a) (b)

Figure 6.6 (a) Examples of classes and properties of the classes account, email and domain that are automatically analyzed (b) Part of a heat map generated from context paths of the Demisto playbook, showing the properties of classes of an ontology. The Y-axis indicates class and the X-axis indicates the properties of a class.

DecOr uses SemOnto to identify properties of a dAPI element generated by SecAPIGen and the relationships between different dAPI elements. SemOnto automates the generation of semantic definitions of different dAPI elements from playbooks' descriptions. It should be noted that playbooks from different vendors might have different structures. To extract features from playbooks with different structures, a human expert (e.g., playbook designer) needs to define the key fields of their playbooks (i.e., description, tasks, inputs and outputs) to SemOnto. SemOnto can be modified to identify the relevant ontological concepts of dAPI elements from different types of playbooks. As the focus of this chapter is mainly on design and automatic generation of dAPIs, this chapter does not show how SemOnto automatically integrates new concepts to update an ontology. However, SemOnto can be modified to fully or partially update an ontology, depending on the features of security tools and playbooks. To do so, first, the concept of the ontology can be identified by analyzing the input and output descriptions of available playbooks. Then SemOnto can integrate new classes or features of classes with an existing ontology. Before integrating the latest concepts, SemOnto must check whether the class exists in an ontology and verify the consistency. Depending

on an organization's preference, the security team can either modify the concepts or use the standard defined or imposed by a SOAR platform.

The integration of new concepts generated through SemOnto with a bootstrapping ontology is two-fold. Firstly, it performs a manual update and secondly, a semi-automated update. In the manual update, SemOnto suggests a form to the experts to fill in when they make any changes in the underlying execution environment. For example, if a new security tool is added, it requests the security tool's name, type, functional capabilities, input, output and runtime environment. For a semi-automated update, SemOnto uses the playbook to interpret the input and output, identify key classes of ontology from the playbook and recommends them to users. It minimizes the overhead of users manually identifying the concepts. Furthermore, upon being verified by users, dAPIs are used to include new concepts and update an ontology. For example, an organization has a new security tool, *Firewall* that they want to integrate. Three integration APIs are used to identify (*identify.SecurityTool(firewall).ontology*) and validate (*validate.securityTool(firewall).ontology*) the concepts of the ontology and further integrate (*integrate.ontology*) them with the bootstrapping ontology (as shown in Figure 6.2(e)).

6.4 EXPERIMENT

This section presents the experimental setups, implementation details and evaluation procedures that we have followed to answer the main Research Questions (RQ) for evaluation.

- **RQ1.** How effective is SecAPIGen in generating and identifying dAPIs for different tasks?
- **RQ2.** Can SemOnto identify the concepts of an ontology from a playbook?
- **RQ3.** How efficient is DecOr in terms of time?

The goal of these RQs is to evaluate the effectiveness and efficiency of AI technologies to generate dAPIs that support a paradigm shift from traditional SOAR solutions. Section 6.4.1 discusses how we have performed the experiments and section 6.4.2 presents the results we obtained.

6.4.1 Data Collection and Tool Implementation

In this section, we describe the data collection, experimental design and tool implementation procedure that we followed for the evaluation of our proposed approach, DecOr. We used a computer as our experiment environment that has the following configuration: Intel(R) Core™ i7-6600U CPU, 8GB RAM, and Window 10 (64-bit).

6.4.1.1 *Task and Object Corpus*

We downloaded the open-source playbooks of Demisto [166, 189]. A playbook of Demisto is written in YAML format. We extracted the descriptions of the available playbooks' tasks with a focus on generating dAPIs. We separated each task's description from the details of each task to build a task corpus (Figure 6.1). We further extracted outputs descriptions from each playbook and extracted the context paths from each output with a focus on generating the properties of dAPI elements from the output context path. We extracted the objects from the context path that were similar to the dAPI elements. Table 6.6 shows the statistics of the data that we collected in this way, which consist of 2000 unique task descriptions, 194 unique descriptions of the playbook, 448 unique context paths and 292 unique objects.

Table 6.6 Statistics of Demisto playbook

Features	Number
Number of playbooks	194
Number of tasks	2000
Number of context paths	448
Number of objects	292

6.4.1.2 *Benchmark for Generation of dAPI*

The results from using NLP to automate the generation of particular information are usually evaluated based on a benchmark data set [202]. Most of the time, these benchmarks are manually annotated by human experts [202-204]. Unfortunately, there was no such corpus for the dAPIs of a SOAR platform that we could use as a reference benchmark. One of the key reasons behind this situation may be that SOAR platform technologies are still in their early stages of development, evaluation and adoption. To the best of our knowledge, this work is the first one to identify the requirements and

design of dAPI for a SOAR platform. That is why we decided to develop the required benchmark based on the textual descriptions of the tasks, e.g., line 92 of Figure 6.1(b) explicitly describes what task it performs and which security tool it uses. The text description of a task can be considered as an input to SecAPIGen. Following that, dAPI can be manually assigned each task which can also be used as a ground truth. As a result, we manually labeled the dAPIs for 147 tasks that were selected from the playbooks' task corpus. We considered each task as a query from an end-user of a SOAR platform and assigned each task only one dAPI. The generated ground truth served as a benchmark for our experiment. In the following section, we explain how we selected a suitable task description to be used as an input text and annotation instruction for labeling the dAPIs manually.

6.4.1.3 *Ground Truth of Experimental Queries, Commands and dAPIs*

We built two sets of ground truth for SecAPIGen: (i) *ground truth 1* to evaluate the generation of dAPIs from the task descriptions (i.e., Algorithm 1) and the identification of dAPIs' parameters (i.e., Algorithm 2) using dependency parsing and (ii) *ground truth 2* to evaluate all the generated dAPI elements (i.e., Algorithm 1 and Algorithm 2) and the identification of semantically similar APIs (i.e., Algorithm 3).

Ground truth 1. We randomly selected a small number of tasks to label the dAPIs manually. The goal is to create experimental queries for our proposed algorithms. We considered a case where a security team will be using DecOr to support security orchestration. Thus, they would already have knowledge about the tasks and incidents being considered. As a result, we assumed that a user would provide queries in imperative sentence form and would not provide any incomplete descriptions. Furthermore, we assumed a single query would be associated with a single task. Hence, the task description should not be in a complex or compound sentence structure [193, 194].

Considering the goal of dAPI is to hide the details of the SOAR's task at different levels, we also assumed each task should be self-explainable and should not refer to an object or element of the previous task. The task descriptions gathered from the playbooks were written in a mixed form. For example, some descriptions were in a question form and some were contained in multiple sentences. Considering the above-mentioned assumptions, we defined the following criteria that selected task descriptions to be

satisfied: (i) each description should explain only a single task; thus, we discarded any task that had a complex or compound sentence structure or clause. An example of a discarded task according to this criterion is "*if you identify suspicious URL, then analyze in-depth in next steps*", (ii) a task should have a complete description; if the first two parts of a dAPI could not be identified from the description, we considered them to be incomplete tasks. (iii) the tasks should be in an imperative sentence form; thus we removed the tasks that were in question or interrogative sentence forms (e.g., "*are there any files to hunt*?"). (iv) A task should not refer to an entity of the previous task. Hence, we removed or modified tasks that were referring to an entity of the previous task. For example, "*block these on the proxy gateway*". If a task contained words such as 'this', 'that', and 'them', then we replaced those words with exact object or entities if they had been identified from previous tasks. Otherwise, we removed those sets of tasks. In this way, we labeled the dAPIs for 147 tasks in total. Table 6.7 shows the average length of the selected tasks is 6 words. An average length of 6 words is acceptable considering the descriptions represent commands that are given to a SOAR platform for a specific task.

Table 6.7 Statistics of ground truth for the generation of dAPI

# of task	Statistics of task length			No. of unique methods and parameters in dAPI				
	Mean	Median	Mode	$\mathbf{a}_{i1}^{\mathbb{D}}$	$\mathbf{a}_{i2}^{\mathbb{D}}$	$\mathbf{a}_{i3}^{\mathbb{D}}$	Param of $\mathbf{a}_{i2}^{\mathbb{D}}$	Param of $\mathbf{a}_{i3}^{\mathbb{D}}$
147	6.5	6	7	17	46	50/76	33/38	19/41

We labeled the dAPI following an annotation instruction which further implemented as the algorithms proposed in sections 6.3.2.1 and 6.3.3.1. The detailed annotation instruction to label a dAPI from a task description is provided in Appendix A1. Any annotator can follow that instruction to design dAPIs for a particular set of tasks. We considered the first part the main method (or dAPI element) of creating a dAPI. Multiple tasks can be performed by a similar method. For example, among the 147 tasks, 23 tasks are performed by the dAPI method *verify* and 11 tasks are performed by *block* (more details in Appendix A2). Table 6.7 shows the number of unique API elements seen in each part of dAPIs in our ground truth. Some dAPIs might not have all the parts. Table

6.7 further shows the number of unique API elements that were seen in each part (x) and the total number of unique API elements (y) in that part in the form of x/y. For example, for parameters of the third part, from 41 unique API elements only 19 were seen in this part and others were also seen in the previous parts (i.e., the second part $a_{i2}^{\mathcal{D}}$, third part $a_{i3}^{\mathcal{D}}$ or a parameter of the second part) of APIs for different tasks. However, as mentioned before each dAPI must have the first two parts, which are $a_{i1}^{\mathcal{D}}$ and $a_{i2}^{\mathcal{D}}$. For example, Table 6.8 shows that the third part i.e., $a_{i3}^{\mathcal{D}}$ of task t_1 is empty and the parameters of $a_{i3}^{\mathcal{D}}$ for a task t_2 are empty. In all the examples in Table 6.8, $\mathbf{a_{i1}^{\mathcal{D}}}$ and $\mathbf{a_{i2}^{\mathcal{D}}}$ are available.

Table 6.8 Examples of ground truth for evaluation of SecAPIGen

#	**Task description**	**dAPI**	$\mathbf{a_{i1}^{\mathcal{D}}}$	$\mathbf{a_{i2}^{\mathcal{D}}}$	$\mathbf{a_{i3}^{\mathcal{D}}}$	**Param of** $\mathbf{a_{i2}^{\mathcal{D}}}$	**Param of** $\mathbf{a_{i3}^{\mathcal{D}}}$
t_1	Block the malicious IP	block.IP(malicious)	block	IP	-	malicious	-
t_2	Block the malicious IP using the firewall	block.IP (malicious). firewall	block	IP	firewall	malicious	-
t_3	Block the malicious IP using checkpoint firewall	block.IP(malicious). firewall(checkpoint)	block	IP	firewall	external	checkpoint

Ground truth 2. We further combined the annotated dAPI elements into a single dAPI element based on their semantics and the task they were performing. For example, *"quarantine the endpoint"* and *"isolate the endpoint"* refer semantically to the same task. With the above annotation, there are two different API elements in the first part: *quarantine* and *isolate* of ground truth 1. To build ground truth 2, we combined these API elements and presented them as a single API element. For example, we used *quarantine* instead of *isolate*. Thus, ground truth 2 refers to tasks that are related to *isolate* and *quarantine* using the API element *quarantine.* Considering $\mathbf{a_{i1}^{\mathcal{D}}}$ and $\mathbf{a_{j1}^{\mathcal{D}}}$ were semantically similar to $\mathbf{a_{k1}^{\mathcal{D}}}$, which were then combined to execute by $a_{k1}^{\mathcal{D}}$, we presented this in the form: $a_{k1}^{\mathcal{D}}$: $(a_{i1}^{\mathcal{D}}, a_{j1}^{\mathcal{D}}, a_{k1}^{\mathcal{D}})$. Appendix A3 shows which dAPI elements are combined into a single dAPI element for the first part in the form of $a_{k1}^{\mathcal{D}}$: $(a_{i1}^{\mathcal{D}}, a_{j1}^{\mathcal{D}}, a_{k1}^{\mathcal{D}})$. After combining semantically similar dAPIs, ground truth 2 had 17 unique dAPI methods in the first part, and 36 in the second part of the APIs. Unlike the first part, for later parts,

the annotated dAPIs tend to have fewer similarities. We used 70% of ground truth 2 as the validation set to build and select the similarity measurement types and similarity scores for Algorithm 3. The remaining 30% was considered as the testing set to evaluate the performance of SecAPIGen, when combining semantically similar APIs.

Two authors independently labeled the dAPIs following the annotation instruction in Appendix A1. For some tasks, the annotator identified the dAPI object elements in the second part, third part and the parameter of these parts in an alternative order. An example of such a task is "*send an sms alert using Twilio*", for which two annotated dAPIs by annotator 1 and annotator 2 were "*send.sms(alert).twilio*" and "*send.alert(sms).twilio*". The annotated dAPIs showed that annotator 1 labeled "*sms*" in the second part and *"alert"* as a parameter of the second part and annotator 2 labeled "*alert*" in the second part and *"sms"* as a parameter of the second part. dAPI "*send.sms(alert).twilio*" will first identify the way to send an object and then send that object; whereas dAPI "*send.alert(sms). twilio*" will first identify the object and then the way to send that object. Execution of both these dAPIs will successfully send an alert to Twilio via SMS. As both of these APIs were fulfilling the objective of a requested task, we considered both of them to be a correct label and used the second form in our ground truth. For some tasks, partial matching was noticed in the parameters of the second and third parts. We considered partial matching to be an agreement between the annotators. We used the Cohen Kappa score [205-207] to measure the agreement between the two labelers after resolving the abovementioned issues. We calculated the agreement between the two annotators for each part of the dAPI elements and also the overall agreement score for annotating all dAPI elements. The overall kappa score was 0.82. Both annotators had 100% agreement on the first part, thus the kappa score for the first part was 1. For the second and third parts, the scores were 0.86 and 0.7 respectively. We noticed the score reduces for identifying parameters of the second and third parts, at 0.68 and 0.6 respectively. Based on the kappa score, the ground truth was considered to have a good reliability score [205-207].

6.4.1.4 ***Evaluation Metrics***

We first evaluated the correctness of DecOr using *precision*, *recall* and *F1-measure*, which are important, classic metrics to evaluate NLP-based models [202, 208, 209].

Precision referred to the ratio of correct dAPI elements and the total number of dAPIs that were generated by SecAPIGen for the corresponding tasks. Recall referred to the proportion of the actual number of dAPIs SecAPIGen successfully generated. To understand precision and recall, we need to first understand the concept of *true positive* and *false positive*. For our experiment, true positive referred to the fact that SecAPIGen generated a dAPI element for a task, and in fact the corresponding dAPI element was recommended for that task in the ground truth. Whereas, false-positive referred to the fact that SecAPIGen generated a dAPI element for a task but in actuality the dAPI element was not recommended for that specific task. Alongside the notions of true positive and false positive, we also considered false negatives. False-negative referred to the cases where SecAPIGen predicted that dAPIs were not available to predict a task, but in fact dAPI elements were available to execute the task. The following are equations for precision and recall in terms of true positive, false positive and false negative.

$$\text{Precision} = \frac{\text{True positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{F1-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 measure is considered a good performance metric because it leverages both precision and recall metrics. We obtained the f1-measure simply by taking a harmonic mean of precision and recall. From the equation of precision and recall, we can see that true positive precision considers only false-positives and recall considers false negatives. Conversely, the f1-measure focuses both on false positives and false negatives.

In addition to the correctness, we considered the average time DecOr took to generate a dAPI from a task description and the average time it took to find similar APIs.

6.4.1.5 *Evaluation Procedure*

Evaluation with Ground Truth (E1): We compared the dAPIs that were generated by SecAPIGen against the two ground truths. As discussed in **section 6.3.3,** SecAPIGen has two modes: novice mode for non-technical users and expert mode for the technical user. In novice mode, users with little knowledge about the available dAPI provide the task

description/query in imperative sentence form. In expert mode, users have knowledge about some of the available dAPIs. Hence, an expert user provides a dAPI to perform a task instead of a task description. As a result, we first experimented with the use case of novice mode using ground truth 1, where SecAPIGen generated the dAPI elements from task descriptions. Next, we conducted the experiment with the use case of expert mode using ground truth 2, where SecAPIGen identified the semantically similar API elements.

Evaluation based on Expertise (E2): To evaluate the identified ontological concepts, we manually selected the generated concepts that had a higher frequency and checked whether they were able to capture information about an object. We also compared the generated concepts with the existing ontology if a match was found, to check the similarities in terms of the identified properties and related classes.

Procedure: Our evaluation includes the following steps:

- Use Algorithm 1 and Algorithm 2 to take inputs in novice mode and generate a dAPI.
- Evaluate the generated dAPI with ground truth 1.
- Use Algorithm 3 to identify the semantically similar dAPIs of the generated dAPI.
- Use 70% of ground truth 2 to build and select the similarity measurement metrics and similarity scores, and 30% as the testing sets for performance evaluation.
- Use Algorithm 4 to identify the concepts of the ontology.
- In evaluation E1, calculate precision, recall and f1-measures for each element of the dAPIs to answer RQ1.
- In evaluation E1, calculate the response time to generate a dAPI from the task description to answer RQ3.
- In evaluation E2, based on human expertise, identify the correctness of the generated concepts and check with respect to the dAPI elements to answer RQ2.

6.4.1.6 *Language and Libraries*

We used spaCy, which is a free open source python library [197] widely used for NLP-related tasks, to find the dependency parse tree for an input task [196, 210, 211]. We developed Algorithm 1 and Algorithm 2 using spaCy. We further used python NTLK wordnet packages for Algorithm 3 to identify the wordnet and synonym sets (synsets) for each word (i.e., the elements of an API) [198-200]. We tried the different similarity

measurement metrics that are provided by wordnet to find similarity measurement metrics types and scores that best suit our experiment and datasets [201, 212]. We further used the python *Networkx* package [213] to automatically draw the classes and properties of the classes through a network graph. We used the network graph to manually evaluate the identified concepts of the ontology.

6.4.2 Results and Analysis

We conducted experiments for the design and generation of dAPI using AI technologies (i.e., NLP and semantics) to answer the three research questions. We evaluated each component of DecOr (i.e., SecAPIGen and SemOnto) separately, which ultimately provided the overall evaluation of DecOr.

6.4.2.1 *RQ1. How effective is SecAPIGen in Generating and Identifying dAPIs for Different Tasks?*

RQ1 mainly focuses on evaluating the effectiveness of SecAPIGen. We ran extensive experiments for the generation of dAPI to answer **RQ1**, which is essentially about the generation of each API element. Based on the results of the experiment, we explored three sub-questions:

- **RQ1.1**. Does dependency parsing help in generating different parts of a dAPI?
- **RQ1.2.** Which similarity functions should we choose for identifying semantically similar words?
- **RQ1.3.** Does SecAPIGen help in identifying the semantically similar API elements that are generated using dependency parsing?

To answer **RQ1.1**, we evaluated the use of dependency parsing in generating different API elements, which mainly evaluated the performance of Algorithm 1 and Algorithm 2. Figure 6.7 shows the performance of SecAPIGen when generating different parts of the dAPIs using dependency parsing. It shows that the precision and recall of SecAPIGen are higher in identifying the first two parts of an API, whilst it has lower precision and recall for identifying the third part of a dAPI. Furthermore, we can see that the precision for identifying the parameters of the second and third API elements is higher than for recall. The reason behind the higher precision is that most of the dAPI elements for the second and third parameters were empty, which the algorithm identified correctly. For

some test cases, the second and third parts of the parameter had multiple dAPI elements. SecAPIGen failed to identify all the dAPI elements for all cases; however, it partially identified them for most cases. Besides considering there were some errors involved in human judgment, the ground truth might not be 100% correct.

We considered two popular similarity functions, Wu-palmer and Resnik, among several available similarity measure functions to answer **RQ1.2** [201, 212]. The Wu-palmer similarity function is a structure-based measure and the Resnik similarity function is an information content-based measure. The similarity score for Wu-palmer (WP) is between 0 and 1 and Resnik (Res) is integer values greater than or equal to zero. We ran Algorithm 3 to consider the different similarity scores of WP and Res to select a suitable value for the similarity score of Algorithm 3. Figure 6.8 shows the results from identifying the first two parts of dAPI elements using both of the similarity metrics with different values of the similarity score. For the later parts of the dAPI elements, there were not many similar API elements that could be combined. As a result, we simply demonstrate the results in Figure 6.8 for the first two parts of the dAPIs.

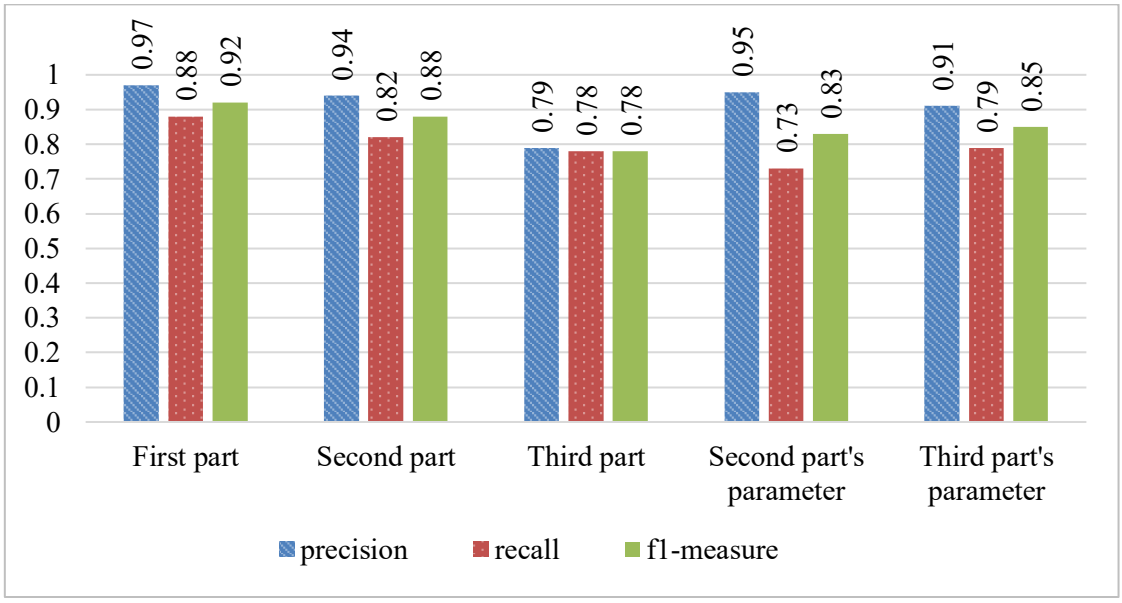
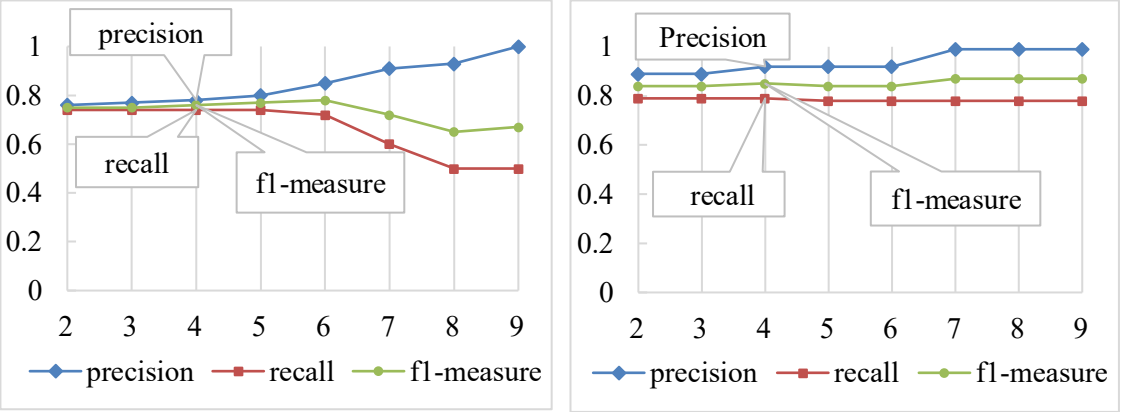


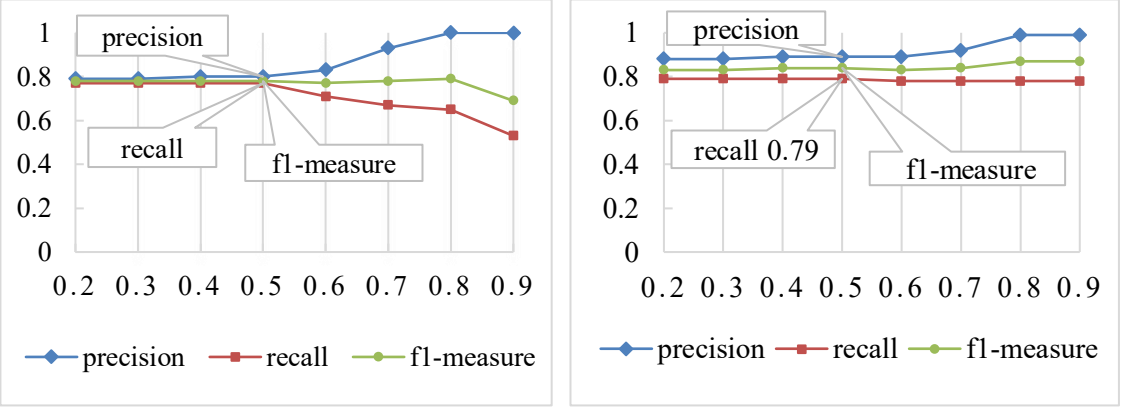
Figure 6.7 Performance of SecAPIGen when generating declarative API elements using dependency parsing

The results of Figure 6.8 reveal that even though Algorithm 3 has higher precision with a higher similarity score, the recall decreases to below 60% with higher similarity scores. We considered the recall value to select the similarity functions and similarity

scores. If the recall were similar, then we considered the F1-measure. The reason for choosing recall is that recall reflects the actual dAPI elements that were correctly identified by the algorithm, where precision considers the ratio of correct dAPI elements that were identified. Even though the F1-measure provides the mean of precision and recall, we did not entirely rely on the F1-measure, because with higher precision and lower recall (e.g., less than 60%), the value of the F1-measure seems to be higher.



(a) Performance of Res for identifying the first part (b) Performance of Res for identifying the second part



(c) Performance of WP for identifying the first part (d) Performance of WP for identifying the second part

Figure 6.8 Performance of SecAPIGen to generate first part and the second part of dAPI using two types of similarity measurements: Resnik (Res) and Wu-palmer (WP) similarity; precision, recall and f1-measures with respect to different RES scores for identifying (a) the first part and (b) the second part of dAPI; precision, recall and f1-measures with respect to different wp similarity scores for identifying (c) the first part and (d) the second part of dAPI

All four graphs in Figure 6.8 show higher precision values (close to 1) with higher similarity scores. Mostly, with higher similarity scores, the dAPI elements that did not have any similar dAPI were identified accurately. However, with higher similarity

scores, semantically similar dAPI elements were not identified, which resulted in lower recall. For example, "*quarantine*" and "*isolate*" were semantically similar API elements that were categorized under the dAPI element "*quarantine*". With a higher similarity score, Algorithm 3 failed to identify "*isolate*" as similar to "*quarantine*". Figure 6.8(a) shows that for the RES similarity function, the value of recall decreases when the similarity score is greater than 5.

Similarly, Figure 6.8(c) shows that the recall of similarity function WP degrades when the similarity score is greater than 0.5. Furthermore, it also shows that the recall for WP degrades when the similarity score is less than 0.3. The f1-measures for WP with similarity scores 0.4 and 0.5 are 0.78. As shown in Figure 6.8(b) and Figure 6.8(d), the performances of both similarity functions are quite similar in identifying the second part of dAPI elements. Both graphs also show steady values of recall and f1-measure. We identified the lack of similar dAPI elements in the second part of dAPI as the reason behind this similar and steady performance. Moreover, the performances in Figure 6.8(b) and Figure 6.8(d) are quite similar to the performance in Figure 6.7, which ultimately reflects the performance for generating dAPI elements without considering semantically similar dAPI elements. By observing the value of the f1-measure and recall for both similarity functions, we considered both as suitable similar functions for identifying semantically similar dAPI. We chose 4 as the similarity score threshold for RES and 0.5 as the similarity score threshold for WP.

After selecting the similarity score, we reported how accurately SecAPIGen generated dAPIs for different tasks with similar dAPI elements to evaluate RQ1.3, which mainly evaluated the overall performance of SecAPIGen. Table 6.9 shows the performance of SecAPIGen when identifying the first and second parts of dAPIs using the testing dataset. The precision, recall and f1-measure for identifying the first part of dAPI are 0.88, 0.8 and 0.84, respectively. Table 6.9 shows the precision, recall and f1-measure for identifying the second part of dAPI are above 88% for Resnik. On the other hand, the precision, recall and f1-measure for identifying the second part of dAPI for Wu-Palmer are above 90%. The results show that SecAPIGen accurately generated dAPIs and identified the semantically similar APIs 80% of the time for the first part and 89% of the time for the second part.

Table 6.9 Performance of SecAPIGen

dAPI element	Similarity function and score	Precision	Recall	F1-measure
First Part	Wu-palmer, 0.5	0.88	0.8	0.84
Second Part	Wu-palmer, 0.5	0.95	0.91	0.93
Second Part	Resnik, 4	0.93	0.89	0.91

6.4.2.2 *RQ2. Can SemOnto Identify the Concepts of an Ontology from the Playbook?*

To evaluate the performance of SemOnto, we first extracted the objects from the context path and then generated the concepts of the ontology following the steps of Algorithm 4. Analyzing 448 *output context paths* and 292 objects, it returned 71 concepts. Among these 71 classes, 37 classes were similar to generated dAPI elements. We further analyzed these classes to evaluate the data properties and object properties associated with each class. For example, considering the ontological concept *file*, we found the object properties (i.e., classes associated with a *file*) and data properties of a *file.* To automate the execution of a task, an ontology of a SOAR platform needs to have these concepts. The analysis of these classes revealed details of the security tools. For example, the identified classes, *Atd*, *Cuckoo*, *Anyrun*, *Checkpointfwrule*, *Joe*, *Wildfire* and *Vmray* refer to *McAfee Advanced Threat Defence*, *Cuckoo Sandbox*, *Checkpoint firewall*, *Joe security sandbox, PaloAlto Wildfire* and *Vmray malware sandbox,* respectively. As we were directly using context paths and each company had its own notation to refer to different attributes in a playbook, some of the class names might not directly match with the name of a dAPI element. Furthermore, we also saw "*Sndbox*" as a class that was referring to a sandbox.

By closely analyzing the name of each class, we removed the above-mentioned ambiguity, while integrating these concepts in an ontology. We further compared these classes with an existing ontology of the SOAR platform and security tools to identify the concepts that were missing. The manual analysis of 71 classes reveals that, for 90% of cases, SemOnto successfully identified the properties of API elements. Thus, recommending the ontological concepts to the security experts will help them to validate the concepts and reduce their burden of manually crafting the concepts of an ontology.

Also, it will help with the cases where the dAPI elements are not identified correctly by SecAPIGen.

We showed that, by analyzing playbooks' inputs and outputs, SemOnto automatically identified the semantic concepts of an ontology, which provided further details about the dAPI elements. The results demonstrate the feasibility of using the playbooks to identify the concepts of an ontology. Using SemOnto, a security team or ontology developer can easily identify the core concepts that are required for the execution of a task. Considering different vendors use playbooks of different types and structures where the tools and tasks also vary, SemOnto helps to gain an overall insight from a playbook about the tools, input and output. As a result, an end-user does not need to learn the structure or libraries of a playbook to incorporate changes that reduce the time taken for manual analysis.

6.4.2.3 ***RQ3. How Efficient is DecOr in Terms of Time?***

To evaluate the efficiency of DecOr, we recorded the *time to generate and identify the similarity of dAPIs*; which is the *query processing time* for SecAPIGen to recommend a dAPI. If SecAPIGen cannot generate a dAPI in a reasonable amount of time, the security team may show no interest in using it at run time to define or update their plans. During the dAPI recommendation phase, given a query, SecAPIGen generated a dAPI element using dependency parsing and then identified synonym sets (synsets) of the generated dAPI element and available dAPI elements to compute the similarity score of two dAPI elements. We considered the time that was required to generate all parts of a dAPI. We recorded the time to generate different elements of a dAPI using dependency parsing and the time to identify semantically similar dAPI elements separately.

Table 6.10 presents the average processing time that SecAPIGen took to generate a dAPI from the task description. It shows that the average time to generate different parts of an API is 9.8 milliseconds (ms), which is the processing time for Algorithm 1 and Algorithm 2. On the other hand, the average time for Algorithm 3 is 160 ms, which is the time taken to identify the semantically similar elements of a dAPI. Algorithm 3 first identified all synsets of two dAPI elements and then performed one to one checking to identify the most similar dAPI element, which resulted in higher processing time, as shown in Table 6.10. It performed the similarity checking, for each part of a dAPI. Hence,

it took a longer time to calculate the semantic similarity of dAPI elements than generating different parts of a dAPI. Overall SecAPIGen took 170 ms on average to generate a dAPI, which included both use of dependency parsing to generate the API elements and then complete the semantics similarity checking. Hence, the semantic framework, SecAPIGen recommends a dAPI in near real-time to the end-user.

Table 6.10. Response times of SecAPIGen for different algorithms

Algorithm	Algorithm 1 and 2	Algorithm 3	SecAPIGen
Average Response Time	9.8ms	160ms	170ms

To sum up, we evaluated DecOr in terms of accuracy and response time to generate dAPIs and dAPI elements using AI technologies. We first showed that using dependency parsing and wordnet, DecOr generated dAPI elements with 90% accuracy, taking only 170 ms on average to generate a dAPI. Furthermore, we showed that SemOnto identified most of the concepts of an ontology that were required to integrate with an existing ontology related to the dAPI elements. Hence, SemOnto contributes to reducing the manual overhead of a security team identifying and crafting the ontologies for the different playbooks provided by different vendors.

6.5 DISCUSSION

6.5.1 Benefits of DecOr

In this section, we discuss the key challenges of the existing SOAR platforms that can be addressed to a certain extent with our proposed dAPI-driven orchestration platform, DecOr.

Wide variety of security tools, technologies and solutions. The first and foremost problem with the adaptability of a SOAR platform is the diversity of security tools with heterogeneous capabilities and requirements. An organization needs to keep pace with the changing threat landscape. For this reason, most organizations end up deploying a wide variety of security tools, each dedicated to a specific set of tasks. Even though a SOAR platform is designed to enable interoperability among security tools, diversity is also found in the SOAR platforms, ranging from plugin-supported to scripts-oriented

(chapter 2). dAPI can easily handle a wide variety of security tools, technologies and solutions. As most of the underlying technologies and tools are abstracted through the three sets of dAPIs, a security team can easily interact with a wide range of security tools and technologies as a result.

Declaration Ambiguity. The commands or the task descriptions in playbooks are defined in natural language (see **Figure 6.1**), whereas an IRP team uses words or phrases they are familiar with or which are popular in their context. For example, consider two tasks N_3 and N_4: *"quarantine the affected endpoint"* and *"isolate malicious host"*. N_3 and N_4 share the same semantics and require similar types of security tools. However, both of these tasks can be found in several playbooks, which are defined by the same company. DecOr handles this sort of ambiguity by identifying these two tasks as similar and provides a single dAPI for both of these tasks.

Technical Expertise. An example is illustrated in **Figure 6.2(a) and (b),** which shows at least three different types of domain expertise required in existing SOAR solutions. However, most organizations have very few security experts. Even so, many organizations are not able to adopt SOARs due to the lack of a dedicated security team or because they have a smaller size of SOC. In a recent study, Gartner predicts that by 2022 even an organization with a security team of five or more will start leveraging SOAR tools for orchestration and automation, where currently it is fewer than 5% [27]. Besides this, the different teams of a SOC require different forms of expertise. As a consequence, instead of building all the required expertise in a single SOC, an organization needs a SOAR that supports the activities of any team and is easy to manage, modify and adapt due to the changing threat landscape. Abstracting the underlying complex architecture through the three sets of dAPI, DecOr can enable a security team to focus on their task alone, without the need to learn the underlying infrastructure.

Manual Creation of Semantics. The existing solutions proposed for enabling semantic integration focus on manually crafting the semantics and defining ontologies to integrate data in a unified way. Extensive domain knowledge is required to create ontologies manually, which makes it difficult for a SOC to keep the ontology up to date. DecOr addresses this issue by identifying the semantics concepts from the playbooks' input and output, which ultimately provide the properties of the security tools and assets

of an organization through which most tasks are performed. Instead of spending the time to craft the semantics concepts, a security team can use the concepts suggested by DecOr and spend time on verifying the concepts and then including them in an ontology.

Proprietary security tools. Of the wide variety of available security tools, most are proprietary. External users have limited access to their API, or vendors publish a set of APIs that other tools or SOAR can use or modify. Developers of SOAR platforms (i.e., designers of playbooks or plugins) need to have extensive knowledge about the usage and constraints of the proprietary security tools. In many cases, several plugins and wrappers of the APIs are designed to integrate the security tools in a SOAR platform. While designing a playbook, a developer also needs to keep in mind the interoperability and compatibility issues associated with different security tools. This issue can easily be addressed if the security vendors expose the properties of their security tools, which can then be integrated easily through the use of integration API, where the playbook developer simply focuses on defining the plans without worrying about the security tools' features.

DecOr aims to address the abovementioned issues by hiding the complexity of the SOAR platform from the security team, so that they can focus on utilizing the SOAR platform to take proactive and informed decisions rather than worrying about underlying security tools, playbooks, diversity in their libraries, configurations and ambiguities in the declarations. We claim that using the dAPI-driven orchestration approach, an organization can easily incorporate different playbooks and security tools into their SOAR platform, thus removing them from being locked into a single vendor. Also, by abstracting their activities through dAPIs, security tools and playbook vendors can ease the path to using multivendor products.

6.5.2 Threat to validity

In this work, we assume each task in the playbooks has only one dAPI through which it can be executed. Nevertheless, there might be multiple dAPIs suitable for executing a single task. In our future work, we plan to extend the design of dAPI, such that each task can be executed by multiple dAPIs. The dAPIs will be recommended to users so they can choose the most suitable dAPI for their respective needs. Furthermore, within the approaches we have used here, several approaches exist to find word similarities [196,

197, 210]. We did not apply all the possible approaches. As a result, there might be some more suitable similarity metrics or approaches that we might have missed. Depending on the datasets used to build the wordnet, the similarity scores can also vary.

To build the used ground truth for SecAPIGen, we only considered imperative sentences with clearly defined task descriptions. We ignored sentences with complex and compound structures because we assume each task description will be given for the execution of a single task. In a practical scenario, this may not always be the case. This issue can be resolved by identifying semantically similar tasks based on sentence similarity, which will identify the semantic similarity between two sentences and thus generate dAPIs for them. Moreover, we have not considered cases where a single dAPI element might have similarities with multiple dAPI elements. We observed cases where a single API element has a similar similarity score with multiple existing dAPI elements.

The developed system presented semantically similar dAPI elements (i.e., two synonymous API elements/words) with one dAPI element even if the tasks are executed by different security tools. The tasks and security tools are mapped with an ontology, which also mapped the user semantics model (i.e., the task defined by the user) with the tools semantic model (i.e., the task performed by the security tool). Though we have not explored the consequences when a user's semantic model cannot be mapped with a task semantic model, this will be an interesting area for future work, to avoid executing tasks that are different from those the user intends to execute.

The experimental results show that the precision of DecOr in generating dAPI elements is lower than 100%. This may result in the wrong execution of a task if the whole process is automated. However, we have designed DecOr to hide the complex internal details of a SOAR platform from its end-user, so that the security team can use it as an intelligence advisory (i.e., a recommendation tool). DecOr is designed to transform a user query into a dAPI and suggests that dAPI to its end-user as an IDE (i.e., python or Java IDE). Depending on the suggested dAPI, the security team can then choose whether to use that dAPI or not. Hence, we consider a precision of lower than 100% is acceptable and applicable in this scenario. The dAPI elements are designed so that they can be mapped with the ontology of a SOAR platform and security tools, which

we believe will help to avoid false execution of tasks by security tools and the SOAR platform.

SemOnto relies on the structure of a playbook's output. Playbooks from different vendors might not follow the same structure, template or notation. However, the purpose of this study is to show that by analyzing an output of a playbook, SemOnto can identify the properties of dAPI elements. A playbook's output also reflects the output of the security tools, so the proposed approach demonstrates the feasibility of using a playbook to identify an API's elements.

SemOnto analyzes the output produced by a playbook and finds the properties associated with a class that must be presented as the core class of an ontology. An existing ontology can also be incorporated during the phase where semantically similar dAPI are identified. Using the combination of ontology and SecAPIGen to identify an API is beyond of the scope of this chapter, which we plan to extend in future work. In this work, we did not show how to automatically use the generated set of dAPI to define the classes of an ontology. Also, the interaction of the three types of dAPI to execute IRPs is not shown in this work and will also be covered in future work.

The evaluation of SemOnto was undertaken manually by one author based on the existing ontology and their domain expertise. This choice may have caused some bias in the evaluation. As the ontologies of the security tools and SOAR platforms are not freely available, it was not possible to perform an automatic comparison of the identified concepts. To minimize the potential impact of the bias of manual analysis, this work can be extended by carrying out a case study where the identified concepts can be shared with domain experts, who can verify the identified concepts as correct or not. The manual validation of the identified concepts of ontologies can be automated by using advanced NLP and Machine Learning (ML) techniques. A future direction to extend SemOnto is to make it fully automated. We plan to extend SemOnto by bringing automation to the manual validation process. One approach to automatically validate the ontological concepts is to update the concepts in an ontology and use that to generate the execution API, running the execution API to invoke a security tool. A fully automated approach can also use the advanced feature extraction techniques of NLP to perform validation of the generated concepts of an ontology.

In this work, we have evaluated DecOr in terms of effectiveness and efficiency. A qualitative evaluation of dAPIs is yet to be performed. We plan to design a study to collect feedback from novice users of a SOAR platform and thereby evaluate the designed dAPIs. The work can also be extended to collect a SOAR platform users' feedback to find the usefulness of dAPI within a changing execution environment.

6.6 RELATED WORK

In this section, we provide an overview of the most relevant research in the area of security orchestration in the light of AI-enabled design and generation of dAPI for a SOAR.

6.6.1 Security Orchestration and Automation Solutions

In chapter 2, we conducted a multivocal literature review on security orchestration. The review identified several strategies for orchestration and automation that are incorporated in the state-of-the-practice and state-of-the-art. Our proposed approach here is partially motivated by the results of this review. The multivocal review showed that even though the existing SOAR incorporates a combination of different automation strategies, the planning and decision-making processes largely rely on human expertise. Security analysts, network administrators, security engineers, forensic teams, incident response teams, a staff of SOC, including a designer, developer and engineer for SOAR are involved in the decision-making and response actions of such a platform. As a consequence, they must be able to interact easily with a SOAR platform to leverage the power of automation and orchestration. However, we noticed little focus from vendors or researchers on making such a solution easily adaptable and modifiable for end-users.

SOAR platform vendors or designers perform pre-assessment of an organizations' security requirements, hardware, software system(s) and available security tools, which form the underlying execution environment of a SOAR platform. The key functionalities of a SOAR platform mainly include the integration, orchestration and automation of security tools' activities and repetitive tasks of security experts. Most commercial SOAR platforms support the integration of the full stack of security tools and information systems (i.e., open-source and proprietary) through APIs, plugins or modules [32, 143, 161, 167, 214]. As most of the security tools are proprietary and each tool has its own

form of API, the integration process is mostly reliant on human experts. A wide range of solutions is seen in security tools' integration. However, the integration of technologies and APIs does not follow a standard that can be followed by end-users to integrate a new set of security tools. Without a well-defined orchestration process, a SOC cannot take the full benefit of integrating existing security tools. The objective is eventually to drive the automated workflow from an incident response plan.

As discussed earlier, an incident response team defines a set of tasks that are required to be executed in response to an incident. These sets of tasks are logically sequenced with chained data flow in a playbook. A playbook contains the automated workflow, which is integrated into a SOAR platform. The majority of practitioners build the workflow based on known use case scenarios. Even though the ultimate goal of a playbook is orchestration and automation, differences are seen in the ways playbooks are built and managed. For example, commercial vendors such as Demisto [35] and Forescout [136] have different forms of playbooks and different ways to interact with them. ForeScout has built a dedicated rule-based engine to automate the workflow [146], whereas Demisto is working toward building a collaborative playbook where anyone can contribute to the design of a playbook. Most of the tools built for workflow design are not suitable for automating the integration of security tools required for task execution. As a result, scripting tools are used to perform automation, which mostly comprise custom codes written by SOAR developers [166, 215]. A developer usually writes scripts to configure a SOAR platform and playbook to integrate security tools in the automated workflow. Execution of such scripts automates a playbook and thus automates a response against an incident.

The continuous changes in the threat landscape require consistent involvement of a SOC team, with designers and engineers of the aforementioned systems even for an instance where changes could be few. For example, after buying a new point of the product, the security team relies on integration tools to connect and integrate the new system into an existing SOAR platform. Moreover, including the new product information in a playbook requires skilled personal in a SOC to write and update corresponding code efficiently. As a consequence, a security team ends up spending a great deal of time learning the underlying libraries and technologies of a SOAR platform

execution environment. Otherwise, the SOC needs to have a dedicated orchestration team to develop the APIs and write the scripts, modifying or updating the workflow across multiple technology paradigms.

Unlike the aforementioned solutions, this chapter has proposed a framework that eases the use of existing SOARs, irrespective of integration tools and vendors. To do so, instead of proposing or developing new tools, we call for a paradigm shift that requires abstraction of the functionalities of the integration, orchestration and automation technologies. Along with abstracting the task of the abovementioned solutions with sets of APIs, we provide a solution to select the correct set of dAPIs for a task easily. This will enable a security team to interact with a SOAR or security tools integrated into a SOAR for different purposes without having an extensive understanding of the underlying technologies.

We assert that the architecture of security orchestration solution needs to be designed in a way that the complex approaches can be integrated easily with the existing SOAR system, such that little involvement of end-users should be required to adopt any new technology (i.e., security tools, APIs, or plugins). However, most of the SOARs are proprietary and designed in an ad-hoc manner, therefore they do not follow any specific architectural styles or patterns. Vendors of SOAR have little consensus among them about where different industries have different requirements for orchestration and automation based on the capabilities of their security team and product requirements. We believe that our proposed framework is a contribution towards tackling the challenges associated with large scale adaption of SOAR platforms.

6.6.2 AI in Automated Interpretation and Integration

Building a fully autonomous process for integrating security tools in a SOAR platform is quite challenging because of the diverse nature of security tools, which are provided by multiple vendors. The use of AI technologies such as semantic knowledge, ontology, NLP and ML is increasingly gaining attention to automate the interpretation and integration of security tools. Chapter 5 proposes semantic representation of the security tools' capabilities and the tasks of IRPs using ontology instead of designing all possible workflows and scripts for integration of security tools. It considered the process of automating the integration of security tools as a combination of subprocesses:

interpretation (interpret the input and output), *selection* (select security tools), *formulation* (formulate the input), *invocation* (invoke a security tool) and *execution* (execute a task). However, the work considered the process as a whole, while changes in one part might require changing the whole process. Furthermore, we consider each subprocess of selection, interpretation, integration, invocation and execution as standalone processes and encapsulate them through a set of APIs, which will enable any security team to easily choose the point where they want to make the changes.

Several studies focus on developing semantic knowledge for formalizing security-related information (e.g., attacks, vulnerabilities, resources and countermeasures) to enable automated interpretation and automation in the identification of relevant incidents [171, 172, 180, 182, 187]. In chapters 4 and 5, we have also focused on incorporating semantic technologies to formalize different types of tools, based on their capabilities to enable interoperability and interpretability among heterogenous tools. The set of ontologies proposed in chapter 4 has enabled a SOAR platform to interpret the task and command of the security tools. Chapter 4 explicitly refers to the involvement of experts during the development of the ontologies. It proposes a semi-automated approach, leveraging NLP and ML techniques to classify the classes of ontologies from the descriptions in incident response plans. Here, the activities are automatically classified to map with the classes of an ontology. The study did not utilize the playbooks, which are mainly structured to identify the concepts of an ontology. The automation here focused on identifying a suitable class of activity from its description. With the same goal of making a SOAR platform adaptable, but with different objectives, one of the foci of our work is on reducing the human effort of learning the details of the semantic technologies by abstracting the interactions across a SOAR platform and its knowledge base.

We further investigate whether the existing automation techniques could be used to identify the concepts of an ontology from the structured documents. The approaches we found fail to satisfy our objective for many reasons. For instances, significant data are not captured in documentation in the existing SOAR platforms. Research has been seen in the area of information extraction and retrieval to automate the identification of ontological concepts, generation of SPARQL from a keyword or text-based query and

annotation of the classes [216-221], which are mostly applicable for the domain of web services or service composition. A SOAR platform deals with the heterogeneous system (i.e., security tools and information systems) which are mostly software systems that are proprietary and open source. Most of the security tools are proprietary and their manuals, guidelines and API documentation are also heterogeneous in nature and vary based on their purpose. Moreover, the documentation details of these systems vary hugely in terms of their structure and content. As a result, to hide the complex details of querying and crafting the concepts of ontologies, we propose to abstract the task to communicate with the semantic knowledge base (i.e., ontologies and query engines) through a set of dAPIs.

6.6.3 AI in API Generation

Recent studies have been seen on the use of AI technologies, mostly NLP, ML and deep learning, for API mining. They recommend increases in the efficiency of software developers [203, 204, 222-224]. Several studies focus on Python or Java-based projects. The purpose of these chapters is to suggest API and API usages based on developer requests by mining APIs from software documentation, GitHub issues and question answering forums or websites. These studies rely on the available documentation from Python and Java and question answering websites' stack overflow. Unfortunately, these sets of studies are not suitable for the security orchestration domain, as a SOAR platform has a diverse set of security tools and each tool has a specific form of APIs and formats, which varies in terms of classes, methods, parameters and so on. As the design and development of a SOAR platform are mostly done in an ad-hoc manner, vendors share little consensus. As previously mentioned, most of the SOAR platforms are proprietary and their documentation and architecture are not available for mining and analysis like open source repositories. Consequently, for a SOAR platform, a standard set of APIs is not available to end-users like Java and Python, for development purposes. Moreover, Security orchestration is still in its early stages and only at the exploration phase, which results in a few queries in question-answering websites like stack overflow. The incident response plans are dynamic, whereby continuous human involvement is required to update ontologies by extracting features of security tools and then defining plans. Similar to the work on API mining, where a system is designed to take both structured and

unstructured queries, we designed DecOr to work with both structured and unstructured queries provided by end-users.

6.7 CHAPTER SUMMARY

This chapter provides an AI-enabled framework, **Dec**larative API-driven **Or**chestration (**DecOr),** that supports security teams to leverage the advanced features of security orchestration platforms without expecting them to know the inherent operational complexities. DecOr is built on top of an existing SOAR platform. The key idea for DecOr is to free security teams from worrying about the underlying libraries, plugins or modules of a SOAR while executing tasks, modifying or updating IRPs, or integrating new security tools. As part of DecOr, there are three sets of *dAPIs*: orchestration API, integration API and execution API. These dAPIs encapsulate the task of a SOAR platform at different levels of abstraction. The dAPIs are required for the easy maintenance of a SOAR. We have also developed a semantic framework, *SecAPIGen*, that leverages existing AI technologies (i.e., NLP technologies, and semantics relationships) to design and generate the dAPIs from the security tasks' descriptions. We further leverage the synonym sets of WordNet, which is the semantics knowledgebase of words, to identify and combine semantically similar words. Finally, we have proposed a semi-automated approach, SemOnto, which uses the playbooks' output to automatically identify the concepts of an ontology (i.e., an AI approach to semantically formalize heterogenous concepts). These concepts are required for a SOAR to automatically interpret the generated dAPIs.

We ran a detailed experiment with a manually curated benchmark of 147 pairs of tasks and dAPIs where the task descriptions are taken from collaborative playbooks from Demisto (i.e., a SOAR platform). We have shown that the dAPIs are automatically generated using dependency parsing, with an average precision higher than 80% and recall higher than 70%. The results of our experiment have demonstrated that semantically similar words are identified with a precision and recall of 88% and 80%, respectively. The analysis of ontological concepts generated by SemOnto further reveals that among the 71 classes, 37 were related to the generated API elements. The reason behind this result is that we did not consider all the tasks in the playbooks and some classes have different names from the API elements generated by SecAPIGen. On

average, DecOr takes 170 milliseconds to generate a dAPI from a task, which is near real-time. Thus, while using DecOr as intelligence assistance or advisory, a SOCs' security team can obtain the corresponding dAPIs and their properties required for task execution instantly. dAPIs free security teams from understanding the underlying libraries and security tools' details. As a result, the team can focus more on analysis of security incidents and preparing response actions.

Chapter 7

Conclusion

Over the past few years, we have witnessed increasing adoption of security orchestration and automation platforms in Security Operation Centers (SOC). Utilization and implementation of a SOAR platform requires skilled and experienced cybersecurity professionals, including (but not limited to) experienced designers and developers, cybersecurity incident response teams, playbook developers and security analysts. A SOAR platform demands collaboration among SOAR vendors and different security teams in a SOC to quickly adapt to the changes in its underlying execution environment and keep updated with the dynamic threat landscape. A SOAR platform should support an increased level of agility and customization to fulfill security teams' requirements. Lack of a comprehensive view and variations in security team requirements leads to ad-hoc design and development of SOAR platforms. These factors pose several further challenges related to seamless integration, managing interoperation and building skilled teams for the incorporation of new tools and technologies.

In this thesis, we have aimed to provide architecture-level support for designing and implementing an easily evolvable human-centric SOAR platform, which enables smooth integration and interoperation of existing and new security tools. We have proposed, implemented and evaluated a layered architecture style for a SOAR platform. We have implemented a set of frameworks and toolsets leveraging AI technologies to realize the proposed architecture. Our findings show that consideration of architectural design decisions can improve the design and development of a SOAR platform. Security orchestration and automation is undeniably the next line of research that requires more attention from cybersecurity researchers and practitioners. We urge the relevant research and industry

communities to come forward to contribute to R&D for developing and evaluating appropriate architectural support for advanced security orchestration and automation solutions.

7.1 FINDINGS AND CONTRIBUTIONS

This section summarizes the significant contributions and the main findings of this thesis.

7.1.1 Understanding of Security Orchestration and the Automation Landscape

This thesis has provided a holistic insight into the current state-of-the-art and state-of-the-practice of security orchestration. Through a Multivocal Literature Review (MLR), which covers 95 key studies, we have presented an overview of a SOAR platform in Chapter 2. We have provided a working definition of security orchestration. A SOAR platform is designed to work as intelligence assistance for human experts, who can benefit from an automated process and make informed decisions proactively**.** Most of the security vendors provide SOAR platforms to build a connection layer for security tools, through which isolated and heterogenous security tools can interoperate with each other and security teams can interact with security tools from a single pane of glass. Repetitive labor-intensive tasks are automated and orchestrated through a SOAR platform, which increases the operational efficiency of security teams.

By analyzing the existing studies, we have identified three key paradigms, unification, orchestration and automation, that drive our perception of security orchestration. We have analyzed the key functionalities and components of a SOAR platform in terms of these paradigms. The results of the analysis further show most of the SOAR platforms leverage six automation strategies and focus on four execution environments. We have provided a taxonomy of security orchestration, analyzing the key techniques, tools and strategies used by practitioners and researchers for the design and development of a SOAR platform. The reported taxonomy has covered five key dimensions: (i) automation strategies, (ii) execution environments, (iii) task execution modes (iv) deployment types and (v) resource types. Based on the research, we have further identified a few open issues in security orchestration practice and research, including the design of an evolvable SOAR platform, integration mechanism,

orchestration process and the requirement for skilled professionals and human involvement in the automation loop.

7.1.2 Layered Architecture for a SOAR platform

We have proposed a layered architecture of a SOAR platform and an associated design space to support architectural design decisions for a SOAR platform in chapter 3. The proposed SOAR platform's architectural design is based on two key functions and five non-functional requirements. The two key functions are (i) that the SOAR is perceived as a unifier or hub and (ii) that is SOAR is recognized as a coordinator or orchestrator. The five NFR include integrability, interoperability, interpretability, usability and modifiability. The proposed six layers are: the user interface, orchestration, semantic, data processing, integration and security tool layers. They help in the design and implementation of a SOAR platform that provides a security team with the desired functionalities, while fulfilling the NFR.

A layered architecture of a SOAR has been designed, separating the concerns of a SOAR platform into different layers by modularizing the functionalities and components. The separation of the semantic layers has helped to achieve interpretability and also enabled semantic interoperability among the security tools and SOAR platforms. Furthermore, the integration layer has been designed to enable security teams to flexibly integrate heterogeneous security tools (i.e., proprietary, open-source, legacy and COTs), based on their preferred integration mechanism (i.e., plugins, scripts, APIs and modules). Employing a layered architectural style has addressed issues with the evolution of SOAR platforms by providing security teams with the flexibility to choose, modify or add preferred components for deployment and evolution of a SOAR platform. We have developed a Proof of Concept (PoC) SOAR platform, STUn (i.e., Security Tool Unifier), giving consideration to the structure of an ad-hoc SOC infrastructure. Observing the impact of automated integration and interpretation of security tools and incident response processes, we have found that consideration of architectural design decisions has improved the SOAR design practices.

7.1.3 Semantic-based Integration Framework

We have devised and implemented a semantic-based integration framework for automatic integration and interpretation of security tools' data and IRPs' activities in chapter 4. The semantic-based integration of security tools has enabled interoperability among isolated, heterogenous and multi-vendor security tools. We have formalized the key concepts of a SOAR platform as: security tools, their functional and non-functional capabilities and the activities of IRPs. We have introduced an ontological model for SOAR platforms that has three main classes: security tools, capabilities and activities. The relationships among these classes have been defined for automatic selection and invocation of security tools in chapters 4 and chapter 5. We have designed the classes of the ontological model following a systematic and structured approach. Furthermore, we have implemented an annotation and prediction module, leveraging popular NLP and ML techniques. The prediction module has automatically classified the activity descriptions of IRPs in an ontology class, enabling automatic adaption to changes in the activity descriptions, and thereby generating new IRPs. It has also removed ambiguity in activity descriptions when different security teams have defined similar tasks in different words.

The ontological model, annotation module, and prediction module form the integration framework that has been used to create an interoperability model. The interoperability model has been designed to automate the execution of an IRP that has been performed by automatically predicting the classes of activities from their descriptions, identifying the security tools needed and invoking those security tools. The feasibility of the proposed prediction model and interoperability model has been evaluated experimentally. We have found that consideration of NLP and ML techniques have enabled automated interpretation of text-based activity descriptions and the execution of IRPs.

7.1.4 Ontology-driven Integration Process

In order to provide further support for automation, we have proposed a novel ontology-driven integration process in chapter 5. We have observed that to adapt the changes in security tools or IRPs, a SOC requires skilled professionals, who are familiar with the underlying libraries and components of a SOAR platform. APIs, plugins or scripts must

be designed and developed by human experts to adopt the changes, which are usually repetitive, time-consuming and error prone. Most SOCs already suffer from a shortage of skilled professionals; hence, the manual and repetitive process may result in further challenges for the organizations interested in leveraging the benefits of a SOAR platform.

We have found that human interventions can be minimized in the integration process by formalizing the input, output, execution environment and functionalities of security tools. We have proposed and designed a set of ontologies to provide the required formal specifications. We have further proposed an approach, OnSOAR, to automate the process for integration of security tools in four stages: (i) interpretation of the incident, (ii) identification of activities, (iii) selection of security tools and (iv) formulation of commands. Furthermore, a set of rules and structured queries have been defined to keep the ontology consistent and retrieve data from it. The proposed approach has been experimentally evaluated and compared with two existing baseline approaches. The evaluation has demonstrated the effectiveness and efficiency of the proposed approach. This research has demonstrated that the proposed approach can contribute towards an integrable, interpretable, interoperable and modifiable SOAR platform.

7.1.5 AI-enabled Declarative API for Security Orchestration and Automation

We have presented an AI-enabled declarative API-driven orchestration approach, namely DecOr, in chapter 6, for a flexible, scalable and easy to modify SOAR platform. DecOr has mainly been designed as part of the abstraction layer of the proposed layered architecture (chapter 3). Thus, the inherent complexity of a SOAR platform has been hidden from its end users using a set of declarative APIs (dAPIs). We have identified three sets of dAPIs: (i) orchestration APIs, (ii) integration APIs and (iii) execution APIs, to encapsulate the activities related to orchestration, integration and execution of the activities of a SOAR platform. Each dAPI (e.g., block. IP (malicious). Firewall (checkpoint)) has been designed to represent a task (e.g., scan, block and correlate), an object (e.g., IP, and endpoint), properties of an object (e.g., malicious IP) and, for some cases, the security tools (e.g., checkpoint firewall and SIEM) that are required to execute the needed task(s). The dAPIs can enable an end-user to interact with a SOAR without having detailed knowledge about the underlying libraries and configurations of a SOAR platform.

We have also developed a semantic framework, SecAPIGen, to automate the design and generation of dAPIs elements from task descriptions (i.e., activity descriptions or user queries). SecAPIGen leverages the NLP tools and techniques, such as dependency parsing and WordNet, to generate different parts of a dAPI and identify semantically similar dAPIs. We have observed that the playbooks of a SOAR platform are mostly structured, which can be utilized to automatically identify the concepts of an ontology and map the elements of dAPIs with the classes and elements of an ontology. Our proposed solution also includes SemOnto, which uses playbooks' output to automatically identify the concepts of a SOAR's ontology (i.e., it is required for semantic integration and interpretation). The concepts generated by SemOnto can enable semantic interpretation of a dAPI's elements. DecOr has been experimentally evaluated with a detailed experiment of 147 pairs of tasks and dAPIs. The experimental results demonstrate that DecOr has successfully generated dAPIs in near real-time with high precision. Our proposed approach has provided support for an easy to learn, interact with and modify SOAR platform. Based on the reported solutions and the results of the evaluation of the proposed solutions, we can assert that the declarative APIs and the utilization of AI technologies in the design and development of a SOAR platform are able to address several of the challenges of a SOAR platform identified in Chapter 2 effectively.

7.2 FUTURE DIRECTIONS

This thesis has made a significant contribution to the growing body of knowledge and technologies (i.e., methods, processes and tools) for designing and evolving a SOAR platform that can support ease of integration and interoperability of different types of security tools. The findings from the reported research also lay the foundation for several future research opportunities in this area. We consider two key potential areas for future research below.

7.2.1 Software Engineering for Security Orchestration and Automation

Despite the widespread adoption of security orchestration technologies and practices in recent years, several engineering and management issues (such as legal issues, trust

management, adaptability, scalability and usability) have mostly been neglected and need to be addressed. Below we discuss some important points for future research:

7.2.1.1 *Empirical evaluation of the proposed architecture and frameworks*

The proposed architecture and toolsets of this thesis have been evaluated through rigorous experimentation using seven open-source, freely-available security tools, a set of IRPs and 189 playbooks from one SOAR vendor. Further research is needed for a large-scale evaluation of the proposed architecture, toolsets and frameworks with multiple proprietary and enterprise editions of security tools, IRPs, playbooks and use case scenarios. Chapter 2 has identified a dearth of solid studies in the security orchestration domain. There is an important need to define evaluation criteria and metrics to empirically evaluate different aspects of a SOAR platform, including the promised functional and non-functional requirements. Further research is also required for developing and applying a framework for evidence-based evaluation of the security orchestration tools and techniques.

In this thesis, we have proposed a paradigm shift for large-scale realization of a SOAR platform. To the best of our knowledge, this thesis is the first step towards encapsulating the tasks of a SOAR platform through a set of dAPIs. SOAR platform vendors, or security tools vendors, may consider abstracting the activities of their products and tools through a set of standard dAPIs and their underpinning AI techniques to empirically understand whether or not the proposed solution can improve the efficiency and productivity of a SOC team who are dealing with a continuously changing threat landscape.

A useful area of empirical exploration could be conducting large-scale mapping of the existing SOAR platforms and IRPs onto their architecture design decisions to generate patterns and hide interactions among the different components across multiple technology paradigms. Such a large-scale evaluation would provide the usefulness and generalizability of the findings of this thesis.

Another potential area of future research is to evaluate the practical value of the findings of this thesis by seeking feedback from practitioners by performing industry-scale evaluations, whose findings could help improve the proposed architecture and toolsets. For such an evaluation, domain experts should be tasked with designing a

suitable ontology for the security tools used to study the amount of efficiency that can be increased with the use of semantic integration.

Another empirical study could invite a security team to use the semantic integration framework to devise an ontology and cause the declarative APIs to interact with a given SOAR for integrating and interoperating security toolsets.

7.2.1.2 *Reference architecture for security orchestration and automation*

Whilst this research has proposed a generic layered architecture that supports security tools' integration in a SOAR platform, there is need to develop and evaluate a reference architecture for the security orchestration domain. A reference architecture often serves as a guideline or starting point for designing concrete architecture for a software system in a particular domain. Emerging large-scale systems that include the integration of a wide variety of software and new technologies such as cloud, fog, IoT, big data and blockchain have benefited from having a reference architecture [225-228]. Hence, an emerging domain like security orchestration will benefit from research efforts aimed at developing a suitable reference architecture for a security orchestration platform. A reference architecture for security orchestration will help define a model to characterize different components of a SOAR platform and the relationships across the different components of the platform. We assert that the design and evaluation of a suitable reference architecture for large-scale realization and materialization of security orchestration platforms is a significant research challenge. There is an important need for close collaboration between industry and academic researchers from the cybersecurity and software engineering domains to develop a reference architecture for security orchestration.

7.2.1.3 *Evaluating the quality and design of incident response process*

Responding to a security incident as soon as possible is an intricate task for security experts. Despite significant efforts over the decades to detect incidents, the response process is still manual and poorly designed. IRPs can be considered as one of the pre-requisites of a SOAR, based on the IRPs' playbook as it is designed and developed. For successful automation and orchestration, it is important to empirically study and understand different aspects of the existing approaches to developing and assessing IRPs. There are no clear evaluation criteria to assess the quality of an IRP. Given an incident,

a SOC should be able to derive how good an IRP is regarding an automated process. There is an urgent need for evaluation metrics that can be used to evaluate and assess the quality of IRPs. Besides this, following a standard template to define the IRPs will hugely benefit a security team's efforts to respond proactively to a security incident. It is necessary to identify the key features of an IRP: for example, the required security tool that can be used, which part can be automated, whom to contact if a critical system has been compromised, what countermeasures to be followed, and so on.

7.2.2 AI and ML for Security Orchestration and Automation

This research has also identified several opportunities for leveraging AI/ML technologies for developing advanced security orchestration and automation solutions ranging from threat management to automated identification of security tools' features. In the following subsections, we first discuss the potential applications of AI and ML to enhance our proposed frameworks and then highlight how a SOAR platform can benefit from AI technologies.

7.2.2.1 *API generation at runtime*

Several opportunities exist to enhance and leverage the proposed framework, DecOr (i.e., Declarative API driven orchestration), reported in chapter 6, by using AI technologies. One way is to consider that multiple declarative APIs (dAPIs) can be made available to execute a single task due to the diversity of security tools and organizational preferences. NLP and AI technologies can be used to recommend suitable dAPI when multiple dAPIs are available to execute a single task. Another potential area for using AI is to automate the semi-automated approach, SemOnto, which identifies the concepts of an ontology (proposed in chapter 6). For this objective, AI technologies can be explored to extract features from existing semantic knowledge bases (i.e., ontologies, RDF, or structure documents), analyzing different playbooks' structures, and use that information to predict or classify similar features from the structured and unstructured documents of a SOAR (e.g., playbook documentation, security tool documentation and so on).

7.2.2.2 *Automated identification of security tools' features*

This thesis has proposed an ontology-driven integration approach (chapter 4), which requires a detailed definition of the features of different security tools and incident

response plans. Without suitable definitions of a security tool's functional and non-functional capabilities, the proposed approach will not be able to perform the abovementioned tasks. A probabilistic learning model can also be designed to automate the integration process, using the ontological model and the existing security tools' configurations to generate the APIs when an exact match is missing. The use of advanced NLP and ML techniques can help to automate the identification of the security tools' features that are required to automate the execution of an IRP. For example, an AI-enabled automation framework could be developed to recommend the features of security tools from their documentation. This can be achieved by using existing NLP techniques (i.e., Word2Vec and word embedding) and the ML/ Deep Learning (DL) model for analysis of security tools' descriptions or documentation. At the core of such a framework will be a security tool features-centric language model that will be built with the existing security tools and ML/DL models.

7.2.2.3 *Automated analysis and recommendations of a security incident response plan*

There is also a need for future research to develop an automated framework that can recommend a set of IRPs based on different characteristics of an incident to security experts. NLP techniques (e.g., word embedding) and ML techniques (e.g., traditional or deep learning models) can be used for analysis of and recommendations for the existing IRPs. There will need to be a development of incident response-centric language models, which can recommend the possible IRPs to the security team for a real-time response. A tool for automatic analysis of IRPs may also help to identify the key features of an IRP and automate the generation of playbooks from IRPs and other, available security tools.

Appendices

Appendix A1. Annotation Instruction

Annotate the different parts of a dAPI element from the task description: The annotator needs to take each task description from the "*raw_file.xls*" and annotate the different parts (i.e., dAPI element) of a dAPI element. Certain things need to be considered.

(a) Each annotated dAPI will have three parts.

(b) The first part, consists of the key abstract functions such as *block, scan, verify* and *detonate*.

(c) The second part provides details about the task and compromises of the object (e.g., *IP* and *capability*) on which the task needs to be performed.

(d) The third part is to identify the specific components (i.e., *endpoint* and *ontology*) on which the task needs to be done or the types of tools (i.e., *firewall*) that need to be used.

(e) The last two parts of a dAPI take parameters that provide more fine-grained detail about a task. For example, the task "*block external IPs in the Firewall*" specifies the types of IPs. The dAPI for the task will be "*block. IP (external). firewall*" where the second part takes *"external"* as an input parameter. Follow the annotation instruction below to label the task with the corresponding dAPI.

(f) The third part and the parameters of the second and third parts of a dAPI might be empty depending on the task description.

Following are the step by step annotation instruction

Step 1: Identify the Part of speech of a sentence from the list of the task description and the relationship between different words of a sentence or build the dependency parse tree.

Step 2: Annotation of the first part of the dAPI

(g) Identify the root word of a sentence and consider the root of a sentence as the first part of a dAPI.

(h) if a word is the nominal subject of the root of a sentence and the part of speech of the word is verb or noun then consider the word as a first part of the.

Step 3: Annotation of the second part of the dAPI

(i) If the root has a nominal subject where the parts of speech of the nominal subject are either verb or noun only then consider the root as a second part of the dAPI

(j) If a word is a direct object of a root consider it as the second part of the dAPI

(k) If a word is a modifier (adverbial modifier or clausal complement) of the root consider the child of the word that is a nominal subject of the word a. If the child is a nominal subject of the word and the part of speech of the child is a verb then consider it as the second part of the dAPI

Step 4: Annotation of the third part of the dAPI

(l) if a word is a modifier (adverbial modifier or clausal complement) of the root consider the child of the word that is a nominal subject of the word a. If a child is a direct object of the word and the part of speech of the child is a verb then consider it as the third part of the dAPI

(m) If a word is a preposition of the root and the part of speech of the word is the adverbial position then consider the child of the word that is a prepositional object of the word a. If the child of the word is a preposition object and its part of speech is a verb or noun consider the child of the word as a third part of the API ii. Consider the modifier of an API element as the parameter of that API element. For example, *block the external IPs in the Firewall,* here *external* is a modifier (i.e., adjective modifier) of *IP*.

(n) Several cases exist where a single object or subject has multiple dependencies, for example, "*send a message to the source user email address*", here the *"source user email"* is the modifier of the object *address* and parameter of the address API element. Identify the multiple modifiers of the dAPI element and consider them as the list of parameters for that part. Example: address (source user email)

For annotation of the parameter of the second part of the dAPI and the parameter of the third part of the API follow

Step 5: Annotation of the parameter of the second and third part of the dAPI

(b) Each object and subject of the second and third part of the dAPI (also referred to as dAPI element) have further modifiers (e.g., adverbial modifier, adjective modifier and noun compound modifier).

Save the file as "annotated_API_AnnotatorName.xls"

Combine the semantically similar words: Some of the annotated dAPI elements can be combined into a single API element based on their semantics and the task they are performing. For example, "*quarantine the endpoint*" and "*isolate the endpoint*" referring to the same task. With the above annotation, there will be two different API elements in the first part: ***quarantine*** and ***isolate***. The job of the annotator here is to combine the API elements and present them with a single API element. For example: use ***quarantine*** instead of *isolate* or vice versa. In the new annotated API element, the API element *quarantine* refers to the task related to isolate and quarantine that is quarantine → (quarantine, isolate)

Quarantine the endpoint: quarantine. endpoint

Isolate the endpoint: quarantine. endpoint

Similarity ***retrieve*** can be used instead of ***acquire***, and ***get***

Instruction:

(o) Take each API element and consider the rest of the API elements and see whether they seem to be the synonym of each other or tends to have a semantically similar meaning in the context of the task.

(p) Provide the API element list that can be combined in the following form and save the file with the name "semanticAPI_AnnotatorName.docs" / "semantics_api_annotatorName.pdf"

- quarantine →(quarantine, isolate)
- retrieve → (acquire, get, retrieve)

Save the new annotated file as "**annotate_Semantic_API_AnnotatorName.xls**".

Appendix A2. Frequent dAPI Elements

Frequent API elements with the frequency of each element in different parts of the dAPI

Frequency of each dAPI element in first part: retrieve: 28, verify: 23, send: 21, set: 16, block: 11, enrich: 9, detonate: 9, query: 6, create: 5, quarantine: 4, close: 3, parse: 3, categorise: 3, download: 2, print: 2, use: 1, disable: 1

Frequency of frequent dAPI elements in second part: file: 14, information: 9, url: 9, report: 9, ips: 8, severity: 7, incident: 7, account: 6, directory: 6, sample: 5, device: 5, email: 4, object: 4, username: 4, input: 4, endpoint: 3, address: 3, operation: 3, list: 3, domain: 3, result: 2, host: 2, classifier: 2, alert: 2, task: 2

***Frequency of frequent dAPI elements in third part*:** integration: 11, sandbox: 5, room: 4, address: 3, high: 3, url: 3, analysis: 3, repository: 3, domain: 3, account: 2, low: 2, siem: 2, firewall: 2, qradar: 2, information: 2, username: 2, service: 2, zscaler: 2, cuckoo: 2, context: 2, crowdstrike: 2, hash: 2

***Frequent API elements for parameter for second part of dAPI*:** incident: 7, active: 5, account: 4, playbook: 4, user: 3, host: 3, cve: 3, logs: 2, external: 2, email: 2, file: 2, md5: 2, malicious: 2

Frequent dAPI elements for parameter of third part: user: 4, mcafee: 4, email: 3, source: 3, carbonblack: 3, war: 3, joe: 3, security: 3, sandbox: 3, network: 2, crowdstrike: 2, falcon: 2, advanced: 2, threat: 2, defence: 2, blacklist: 2

Most frequent dAPI elements (i.e., objects) except the first part in ground truth: file: 18, incident: 15, account: 12, url: 12, information: 11, report: 11, integration: 11, ips: 10, email: 9, severity: 8, sandbox: 8, directory: 7, username: 7, user: 7, address: 6, host: 6, device: 6, playbook: 6, domain: 6, active: 6, endpoint: 5, sample: 5, input: 5, crowd strike: 5, object: 4, operation: 4, md5: 4, cve: 4, room: 4, source: 4, mcafee: 4, hash: 3, list: 3, task: 3, high: 3, blacklist: 3, analysis: 3, repository: 3, network: 3, carbon black: 3

Appendix A3. Semantically Similar API Element

Following we show the API elements that are considered under an API element for the first part of an API.

retrieve: (acquire, get, retrieve), categorize: (categorize, classify), enrich: (add, enrich), quarantine: (isolate, quarantine), send: (dump, submit, send, return, upload), set: (assign, change, initiate, poll, set), verify: (check, review, verify), block, close, create, detonate, disable, download, parse, print, query, use

Appendix A4. Identified Concepts of an Ontology that are dAPI Elements

Class: account, analysis, anyrun, atd, bitdam, cuckoo, cve, email, endpoint, file, hash, ip, joe, log, malicious, nexpose, qradar, report, result, sample, search, task, threat, url, user.

References

[1] Deloitte and MAPI, *Cyber risk in advanced manufacturing,* [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/manufacturing/us-manu-cyber-risk-in-advanced-manufacturing.pdf.

[2] Verizon, 2020 Data Breach Investigations Report, 2020, [Online]. Available: https://enterprise.verizon.com/en-au/resources/reports/dbir/.

[3] F. Ullah, M. Edwards, R. Ramdhany, R. Chitchyan, M. A. Babar, and A. Rashid, Data exfiltration: A review of external attack vectors and countermeasures, *Journal of Network and Computer Applications,* vol. 101, pp. 18-54, 2018.

[4] K. L. Offner, E. Sitnikova, K. Joiner, and C. R. MacIntyre, Towards understanding cybersecurity capability in Australian healthcare organisations: a systematic review of recent trends, threats and mitigation, *Intelligence and National Security,* vol. 35, no. 4, pp. 556-585, 2020.

[5] F. Luh and Y. Yen, Cybersecurity in Science and Medicine: Threats and Challenges, *Trends Biotechnol.,* 2020.

[6] Z. El-Rewini, K. Sadatsharan, D. F. Selvaraj, S. J. Plathottam, and P. Ranganathan, Cybersecurity challenges in vehicular communications, *Vehicular Communications,* vol. 23, p. 100214, 2020.

[7] A. Hassanzadeh *et al.*, A Review of Cybersecurity Incidents in the Water Sector, *J. Environ. Eng.,* vol. 146, no. 5, p. 03120003, 2020.

[8] C. Ventures. *2019 Official Annual Cybercrime Report*, 2019.

[9] S. Morgan, (2019), *Global Cybersecurity Spending Predicted To Exceed $1 Trillion From 2017-2021,* [Online]. Available: https://cybersecurityventures.com/cybersecurity-market-report/.

[10] ACSC, (July 10, 2020), *COVID-19 malicious cyber activity,* [Online]. Available: https://www.cyber.gov.au/acsc/view-all-content/alerts/covid-19-malicious-cyber-activity#:~:text=The%20malicious%20COVID%2D19%20websites,in%20order%20to%20generate%20profit.

[11] K. Ho, (July 10, 2020), *Australia targeted by sophisticated cyber attacks,* [Online]. Available: https://infrastructuremagazine.com.au/2020/06/23/australia-targeted-by-sophisticated-cyber-attacks/.

[12] D. Maguire, (July 10, 2020), *What we know about the 'sophisticated, state-based' cyber attack on Australia,* [Online]. Available: https://www.abc.net.au/news/2020-06-19/cyber-attack-no-australian-government-organisations-explained/12373190.

[13] C. P. Crowley, John, The Definition of SOC-cess? SANS 2018 Security Operations Center Survey, SANS Institute, August 2018 2018.

[14] Kaspersky, (June 19, 2019), *Return on security investment: internal SOCs halve the financial impact of enterprise data breaches,* [Online]. Available: https://www.kaspersky.com/about/press-releases/2019_internal-socs-halve-the-financial-impact-of-enterprise-data-breaches.

[15] McAfee, (July 10, 2020), *What Is a Security Operations Center (SOC)? ,* [Online]. Available: https://www.mcafee.com/enterprise/en-au/security-awareness/operations/what-is-soc.html#:~:text=A%20Security%20Operation%20Center%20(SOC,and%20responding%20to%20cybersecurity%20incidents.

[16] M. Ahmed, A. Naser Mahmood, and J. Hu, A survey of network anomaly detection techniques, *Journal of Network and Computer Applications,* vol. 60, pp. 19-31, 2016.

[17] Komand, (September 23, 2017), [Online]. Available: https://www.komand.com/.

[18] Intel Security, (October 20, 2017), *Automating the Threat Defence Lifecycle,* [Online]. Available: https://www.mcafee.com/au/solutions/orchestration.aspx.

[19] HEXADITE, (August 07), [Online]. Available: https://www.hexadite.com/.

[20] McAfee, (October 20), *MacAfee Orchestration Platform,* [Online]. Available: https://www.mcafee.com/au/solutions/orchestration.aspx

[21] S. Luo and M. B. Salem, "Orchestration of software-defined security services," in *2016 IEEE International Conference on Communications Workshops, ICC 2016*, Kuala Lumpur, Malaysia, 2016, Conference Paper.

[22] C. Islam, M. A. Babar, and S. Nepal, A Multi-Vocal Review of Security Orchestration, *ACM Computing Surveys (CSUR),* vol. 52, no. 2, p. 37, April 2019.

[23] E. Feitosa, E. Souto, and D. H. Sadok, An orchestration approach for unwanted Internet traffic identification, *Computer Networks,* Article vol. 56, no. 12, pp. 2805-2831, 2012.

[24] Komand, (21/10/2017), *Security automation best practice,* [Online]. Available: https://www.komand.com/.

[25] $(ISC)^2$, Strategies for Building and Growing Strong Cybersecurity Teams, in CYBERSECURITY WORKFORCE STUDY, 2019, 2019 2019, [Online]. Available: https://www.isc2.org/Research/2019-Cybersecurity-Workforce-Study#.

[26] D. VIB, The state of SOAR report, 2019, in The Third annual state of incident response report, Demisto: A Palo Alto networks' company, 2019, [Online]. Available: https://www.paloaltonetworks.com/content/dam/pan/en_US/assets/pdf/demisto/Stat-of-SOAR-Report-2019.pdf.

[27] Gartner, Market Guide for Security Orchestration, Automation and Response Solutions, 2019, [Online]. Available: https://www.gartner.com/doc/reprints?id=1-1OA93LZ7&ct=190716&st=sb.

[28] McAfee, (10/20/2017, 2017), *MacAfee Orchestration Platform,* [Online]. Available: https://www.mcafee.com/au/solutions/orchestration.aspx

[29] FireEye, (November 21), *Security orchestration - best practice for any organization* [Online]. Available: https://www.fireeye.com/solutions/security-orchestrator/wp-best-practices-in-orchestration.html.

[30] I. Security, (October 23, 2017), *Open Security Controller: A security Orchestration Platform for the software-defined datacentre,* [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/open-security-controller-datasheet.pdf.

[31] alienvault, (November 23, 2017), *Security Automation & Orchestration* [Online]. Available: https://www.alienvault.com/solutions/security-automation-and-orchestration.

[32] Swimlane, (November 20, 2017), *Security Automation and Orchestration,* [Online]. Available: https://swimlane.com/use-cases/security-orchestration-for-automated-defense/.

[33] LogRhythm, (November 20), *Security Automation and Orchestration, Respond to Incidents in Seconds—Not Days,* [Online]. Available: https://logrhythm.com/solutions/security/security-automation-and-orchestration/.

[34] Demisto, (October 20, 2017), *Collaborative and Automated Security Operations - A comprehensive Incident Management Platform,* [Online]. Available: https://www.demisto.com/.

[35] Demisto, (December 5, 2017), *Security orchestration and automation,* [Online]. Available: https://www.demisto.com/wp-content/uploads/2017/04/MH-Demisto-Security-Automation-WP.pdf

[36] WHO, (2020), *WHO reports fivefold increase in cyber attacks, urges vigilance,* [Online]. Available: https://www.who.int/news-room/detail/23-04-2020-who-reports-fivefold-increase-in-cyber-attacks-urges-vigilance.

[37] E. Richardson and J. Mahle, (July 10, 2020), *Cyberattacks on the rise during the Covid-19 pandemic,* [Online]. Available: https://www.bizjournals.com/cincinnati/news/2020/06/01/cyberattacks-on-the-rise-during-covid-19.html.

[38] ACSC, (July 10, 2020), *Advanced Persistent Threat (APT) actors targeting Australian health sector organisations and COVID-19 essential services,* [Online]. Available: https://www.cyber.gov.au/acsc/view-all-content/advisories/advisory-2020-009-recommendations-mitigate-apt-actors-targeting-health-sector-and-covid-19-essential-services.

[39] D. Grober, (July 10, 2020), *Roundup: COVID-19 pandemic delivers extraordinary array of cybersecurity challenges,* [Online]. Available: https://www.zdnet.com/article/roundup-the-coronavirus-pandemic-delivers-an-array-of-cyber-security-challenges/.
[40] ACSC, (July 10, 2020), *Threat update: COVID-19 malicious cyber activity 20 April 2020,* [Online]. Available: https://www.cyber.gov.au/acsc/view-all-content/advisories/threat-update-covid-19-malicious-cyber-activity-20-april-2020.
[41] AustCyber, (June 18, 2020), *SCP - Chapter 1 - The global outlook for cyber security,* [Online]. Available: austcyber.com/resources/sector-competitiveness-plan/chapter1.
[42] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *2004 Australian Software Engineering Conference. Proceedings.*, 2004, pp. 309-318.
[43] M. A. Chauhan, M. A. Babar, A. Wasowski, C. W. Probst, and R. Bahsoon, *Foundations for Tools as a Service Workspace: A Reference Architecture*. IT University of Copenhagen, Software and Systems Section, 2016.
[44] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, USA, 2005.
[45] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.
[46] V. Garousi, M. Felderer, and M. V. M, "The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, Limerick, Ireland, 2016.
[47] G. Deepa and P. S. Thilagam, Securing web applications from injection and logic vulnerabilities: Approaches and challenges, *Information and Software Technology,* vol. 74, pp. 160-180, 2016.
[48] M. Leitner and S. Rinderle-Ma, A systematic review on security in Process-Aware Information Systems – Constitution, challenges, and future directions, *Information and Software Technology,* vol. 56, no. 3, pp. 273-293, 2014.
[49] Verizon, (December 5, 2017), *Verizon 2017 Data Breach Investigations Report,* [Online]. Available: http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/.
[50] S. IBM, (December, 2017), *2017 Cost of Data Breach Study: Global Overview,* [Online]. Available: https://www.ibm.com/security/data-breach.
[51] Equifax, (November 10), [Online]. Available: https://www.equifax.com/personal/.
[52] W.-M. Ethan, (January 5, 2017), *Equifax: The company that screwed consumers the most in 2017,* [Online]. Available: https://finance.yahoo.com/news/equifax-company-screwed-consumers-2017-163011368.html.
[53] M. Lee, (December 5, 2017), *Equifax Data Breach Impacts 143 Milion Americans,* [Online]. Available: https://www.forbes.com/sites/leemathews/2017/09/07/equifax-data-breach-impacts-143-million-americans/#1abcaed0356f.
[54] H. Todd, (October 30, 2017), *Credit reporting firm Equifax says data breach could potentially affect 143 million US consumers,* [Online]. Available: https://www.cnbc.com/2017/09/07/credit-reporting-firm-equifax-says-cybersecurity-incident-could-potentially-affect-143-million-us-consumers.html.
[55] P. L. Goethals and M. E. Hunt, A review of scientific research in defensive cyberspace operation tools and technologies, *Journal of Cyber Security Technology,* vol. 3, no. 1, pp. 1-46, 2019.
[56] B. Schneier, (April 30, 2017), *Security Orchestration for an Uncertain World,* [Online]. Available: https://securityintelligence.com/security-orchestration-for-an-uncertain-world/.
[57] M. Dave and V. Viswanathan, (November 20, 2017), *Open Security Controller: Security Orchestration for OpenStack,* [Online]. Available: https://www.rsaconference.com/events/us17/agenda/sessions/6582-open-security-controller-security-orchestration-for.
[58] E. Digiambattista, Enterprise level security orchestration, US Patent 2017/0017795 A1, 2017.
[59] H. Nadkarni, Security orchestration framework, US Patent 9,807,118, 2017.

[60] O. Rochford, (October 29, 2017), *When is Security Automation and Orchestration a Must-Have Technology? – Addressing Gartner's SOAR Question,* [Online]. Available: https://www.dflabs.com/blog/when-is-security-automation-and-orchestration-a-must-have-technology-addressing-gartner-soar-question/.

[61] SWIMLANE, (20/10/2017), *Security Automation and Orchestration (SAO) Capabilities,* [Online]. Available: https://swimlane.com/ebook-sao-capabilities/.

[62] M. Spanbauer, (October 21, 2015), *Security Orchestration – Integration, Process, and Wise Investments Driven by a Security Conductor,* [Online]. Available: https://www.nsslabs.com/blog/analyst-insights/security-orchestration-integration-process-and-wise-investments-driven-by-a-security-conductor/.

[63] J. TRULL, (September 5, 2017), *Top 5 best practices to automate security operations,* [Online]. Available: https://cloudblogs.microsoft.com/microsoftsecure/2017/08/03/top-5-best-practices-to-automate-security-operations/.

[64] I. Resilient, (October 28), *Security Module,* [Online]. Available: https://www.resilientsystems.com/our-platform/incident-response-security/.

[65] TDG, (October 25, 2017), *Security Orchestration Fine-Tunes the Incident Response Process,* [Online]. Available: https://www.turremgroup.com/security-orchestration-fine-tunes-the-incident-response-process/.

[66] Verizon, (September 2, 2016), *Verizon 2016 Data Breach Investigations Report,* [Online]. Available: http://www.verizonenterprise.com/verizon-insights-lab/dbir/.

[67] BakerHosteller, (August 20, 2017), *Be Compromise Ready: Go Back to the Basics - 2017 Data Security Incident Response Report,* [Online]. Available: https://www.bakerlaw.com/events/webinar-be-compromise-ready-go-back-to-the-basics.

[68] l. Dave, (September 20, 2016), *Food chain Wendy's hit by massive hack,* [Online]. Available: http://www.bbc.com/news/technology-36742599.

[69] KrebsonSecurity, (November 20, 2016), *Wendy's Breach,* [Online]. Available: https://krebsonsecurity.com/tag/wendys-breach/.

[70] O. Rochford and P. E. Proctor, Innovation Tech Insight for Security Operations, Analytics and Reporting, Gartner 2015, [Online]. Available: https://www.gartner.com/doc/3166239/innovation-tech-insight-security-operations, Accessed on: October 21, 2017.

[71] I. Security, (October 23), *Open Security Controller: A security Orchestration Platform for the software-defined datacentre,* [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/open-security-controller-datasheet.pdf.

[72] T. Koyama, B. Hu, Y. Nagafuchi, E. Shioji, and K. Takahashi, Security orchestration with a global threat intelligence platform, *NTT Technical Review,* Article vol. 13, no. 12, 2015.

[73] R. Poornachandran, S. Shahidzadeh, S. Das, V. J. Zimmer, S. Vashisth, and P. Sharma, Premises-aware security and policy orchestration, US Patent 14/560,141, 2016.

[74] T. Yu, S. K. Fayaz, M. Collins, V. Sekar, and S. Seshan, "PSI: Precise Security Instrumentation for Enterprise Networks," in *In Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2017.

[75] E. Tom, A. Aurum, and R. Vidgen, An exploration of technical debt, *Journal of Systems and Software,* vol. 86, no. 6, pp. 1498-1516, 2013/06/01/ 2013.

[76] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, Systematic literature reviews in software engineering – A systematic literature review, *Information and Software Technology,* vol. 51, no. 1, pp. 7-15, 2009.

[77] V. Garousi, M. Felderer, and T. Hacaloğlu, Software test maturity assessment and test process improvement: A multivocal literature review, *Information and Software Technology,* vol. 85, no. Supplement C, pp. 16-42, 2017.

[78] V. Garousi and M. V. Mäntylä, When and what to automate in software testing? A multi-vocal literature review, *Information and Software Technology,* vol. 76, pp. 92-117, 2016.

[79] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," in *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*: sn, 2007.

[80] M. Shahin, P. Liang, and M. A. Babar, A systematic review of software architecture visualization techniques, *Journal of Systems and Software,* vol. 94, pp. 161-185, 2014.
[81] M. Zahedi, M. Shahin, and M. Ali Babar, A systematic review of knowledge sharing challenges and practices in global software development, *International Journal of Information Management,* vol. 36, no. 6, Part A, pp. 995-1019, 2016.
[82] S. A. Zonouz, R. Berthier, H. Khurana, W. H. Sanders, and T. Yardley, Seclius: An Information Flow-Based, Consequence-Centric Security Metric, *IEEE Transactions on Parallel and Distributed Systems,* vol. 26, no. 2, pp. 562-573, 2015.
[83] T. Kenaza and M. Aiash, "Toward an Efficient Ontology-Based Event Correlation in SIEM," in *The 7th International Conference on Ambient Systems, Networks and Technologies*, Madrid, Spain 2016, Conference Paper.
[84] H. T. Elshoush and I. M. Osman, "Reducing false positives through fuzzy alert correlation in collaborative intelligent intrusion detection systems - A review," in *International Conference on Fuzzy Systems*, Barcelona, Spain, 2010: IEEE, pp. 1-8.
[85] M. Kamal, A. J. Davis, J. Nabukenya, T. V. Schoonover, L. R. Pietron, and G. j. D. Vreede, "Collaboration Engineering For Incident Response Planning: Process Development and Validation," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, Waikoloa, HI, USA, Jan. 2007, 2007.
[86] R. Saad, F. Nait-Abdesselam, and A. Serhrouchni, "A collaborative peer-to-peer architecture to defend against DDoS attacks," in *2008 33rd IEEE Conference on Local Computer Networks (LCN)*, Montreal, Que, Canada, 14-17 Oct. 2008, 2008.
[87] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah, "Acing the IOC Game: Toward Automatic Discovery and Analysis of Open-Source Cyber Threat Intelligence," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 2016.
[88] C. Fung, Q. Zhu, R. Boutaba, and T. Basar, "SMURFEN: a system framework for rule sharing collaborative intrusion detection," in *Proceedings of the 7th International Conference on Network and Services Management*, Paris, France, 2011.
[89] A. Poller, S. Turpe, and K. Kinder-Kurlanda, "An Asset to Security Modeling?: Analyzing Stakeholder Collaborations Instead of Threats to Assets," in *Proceedings of the 2014 New Security Paradigms Workshop*, Victoria, British Columbia, Canada, 2014.
[90] T. Koyama, K. Hato, H. Kitazume, and M. Nagafuchi, Resilient security technolgoy for rapid recovery from cyber attacks, *NTT Technical Review,* vol. 12, p. 6, July 7 2014.
[91] B. Jaeger, Security Orchestrator: Introducing a Security Orchestrator in the Context of the ETSI NFV Reference Architecture, pp. 1255-1260, 2015.
[92] C. Sillaber, C. Sauerwein, A. Mussmann, and R. Breu, "Data Quality Challenges and Future Research Directions in Threat Intelligence Sharing Practice," in *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*, Vienna, Austria, 2016.
[93] X. Pan, V. Yegneswaran, Y. Chen, P. Porras, and S. Shin, "HogMap: Using SDNs to Incentivize Collaborative Security Monitoring," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, New Orleans, Louisiana, USA, 2016.
[94] S. Chen, C. Guo, X. Yuan, F. Merkle, H. Schaefer, and T. Ertl, "OCEANS: online collaborative explorative analysis on network security," in *Proceedings of the Eleventh Workshop on Visualization for Cyber Security*, Paris, France, 2014.
[95] M. O. Kalinin, "Permanent protection of information systems with method of automated security and integrity control," in *Proceedings of the 3rd international conference on Security of information and networks*, Taganrog, Rostov-on-Don, Russian Federation, 2010.
[96] H. Gascon, B. Grobauer, T. Schreck, L. Rist, D. Arp, and K. Rieck, "Mining Attributed Graphs for Threat Intelligence," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, Scottsdale, Arizona, USA, 2017.
[97] N. Afzali Seresht and R. Azmi, MAIS-IDS: A distributed intrusion detection system using multi-agent AIS approach, *Engineering Applications of Artificial Intelligence,* Article vol. 35, pp. 286-298, 2014.

[98] W. Zhao and G. White, "Designing a Formal Model Facilitating Collaborative Information Sharing for Community Cyber Security," in *2014 47th Hawaii International Conference on System Sciences*, Waikoloa, HI, USA, 6-9 Jan. 2014, 2014.
[99] K. Alsubhi, I. Aib, J. Francois, and R. Boutaba, "Policy-Based Security Configuration Management, Application to Intrusion Detection and Prevention," in *2009 IEEE International Conference on Communications*, Dresden, Germany, 14-18 June 2009, 2009.
[100] W. Zhao and G. White, "A collaborative information sharing framework for Community Cyber Security," in *2012 IEEE Conference on Technologies for Homeland Security (HST)*, Waltham, MA, USA, 13-15 Nov. 2012, 2012.
[101] M. Sourour, B. Adel, and A. Tarek, "Collaboration between Security Devices toward improving Network Defense," in *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, Portland, OR, USA, 14-16 May 2008, 2008.
[102] A. Sadighian, S. T. Zargar, J. M. Fernandez, and A. Lemay, "Semantic-based context-aware alert fusion for distributed Intrusion Detection Systems," in *2013 International Conference on Risks and Security of Internet and Systems (CRiSIS)*, La Rochelle, France, 23-25 Oct. 2013, 2013.
[103] A. Modi *et al.*, "Towards Automated Threat Intelligence Fusion," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, Pittsburgh, PA, USA, 1-3 Nov. 2016, 2016.
[104] X. Chen, B. Mu, and Z. Chen, "NetSecu: A Collaborative Network Security Platform for In-network Security," in *2011 Third International Conference on Communications and Mobile Computing*, Qingdao, China, 18-20 April 2011, 2011.
[105] H. W. Njogu, L. Jiawei, J. N. Kiere, and D. Hanyurwimfura, A comprehensive vulnerability based alert management approach for large networks, *Future Generation Computer Systems,* Article vol. 29, no. 1, pp. 27-45, 2013.
[106] E. Bou-Harb, M. Debbabi, and C. Assi, "Behavioral analytics for inferring large-scale orchestrated probing events," in *Proceedings - IEEE INFOCOM*, 2014, pp. 506-511.
[107] R. Werlinger, K. Hawkey, D. Botta, and K. Beznosov, Security practitioners in context: Their activities and interactions with other stakeholders within organizations, *International Journal of Human Computer Studies,* Article vol. 67, no. 7, pp. 584-606, 2009.
[108] F. D'Aubeterre, R. Singh, and L. Iyer, A Semantic Approach to Secure Collaborative Inter-Organizational eBusiness Proceses (SSCIOBP), *Journal of the Association of Information Systems,* Article vol. 9, no. 3-4, pp. 231-266, 2008.
[109] A. J. Varela-Vaca and R. M. Gasca, Towards the automatic and optimal selection of risk treatments for business processes using a constraint programming approach, *Information and Software Technology,* Article vol. 55, no. 11, pp. 1948-1973, 2013.
[110] E. Al-Shaer, X. Ou, and G. Xie, *Automated security management*: Springer International Publishing, 2013, pp. 1-187. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84948111330&doi=10.1007%2f978-3-319-01433-3&partnerID=40&md5=331858d13f0a89fa7babffcd92eeaf58.
[111] R. A. Jones and B. Horowitz, A System-Aware cyber security architecture, *Systems Engineering,* Article vol. 15, no. 2, pp. 225-240, 2012.
[112] F. Tanemo, I. Hayashi, M. Tanikawa, and T. Abe, Tighter security operations to help provide brands that are safer and more secure, *NTT Technical Review,* Review vol. 10, no. 10, 2012.
[113] J. S. Li, C. J. Hsieh, and H. Y. Lin, A hierarchical mobile-agent-based security operation center, *International Journal of Communication Systems,* Article vol. 26, no. 12, pp. 1503-1519, 2013.
[114] T. Sadamatsu, Y. Yoneyama, and K. Yajima, Practice within fujitsu of security operations center: Operation and security dashboard, *Fujitsu Scientific and Technical Journal,* Article vol. 52, no. 3, pp. 52-58, 2016.
[115] R. Floodeen, J. Haller, and B. Tjaden, "Identifying a shared mental model among incident responders," in *Proceedings - 7th International Conference on IT Security Incident Management and IT Forensics, IMF 2013*, Nuremberg, Germany, 2013, Conference Paper.
[116] T. Ntouskas, G. Pentafronimos, and S. Papastergiou. *STORM - Collaborative security management environment, Lecture Notes in Computer Science (including subseries Lecture*

Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6633 LNCS, pp. 320-335, 2011.

[117] K. Jeong, J. Park, M. Kim, and B. Noh, "A security coordination model for an inter-organizational information incidents response supporting forensic process," in *Proceedings - 4th International Conference on Networked Computing and Advanced Information Management, NCM 2008*, Gyeongju, South Korea, 2008, vol. 2: IEEE, pp. 143-148.

[118] FireEye, (October 31, 2017, 2017), *Security Orchestrator: Simplify threat response through integration and automation,* [Online]. Available: https://www.fireeye.com/solutions/security-orchestrator.html.

[119] IBM, (August 20), *IBM Resilient,* [Online]. Available: https://www.resilientsystems.com/.

[120] J. Blankership, S. Balaouras, B. Barringham, and R. Birrell, (October 10, 2017), *Breakout Vendors: Security Automation And Orchestration (SAO),* [Online]. Available: https://www.forrester.com/report/Breakout+Vendors+Security+Automation+And+Orchestration+SAO/-/E-RES136903.

[121] D. Forte, (September, 2017), *Security orchestration & Automation: parsing the Options,* [Online]. Available: https://www.darkreading.com/threat-intelligence/security-orchestration-and-automation-parsing-the-options/a/d-id/1329886?piddl_msgid=329392.

[122] THREATCONNECT, (October 20), *Security Automation and Orchestration,* [Online]. Available: https://www.threatconnect.com/security-automation-orchestration/.

[123] R. a. Market, (November 4, 2016), *Security Orchestration Market by Component (Solution and Service), Application (Threat Intelligence, Network Forensics, Ticketing Solutions, and Compliance Management), Deployment Mode, End User, and Vertical, Region - Global Forecast to 2021,* [Online]. Available: https://www.researchandmarkets.com/research/jcmnbx/security.

[124] alienvault, (November 23), *Security Automation & Orchestration* [Online]. Available: https://www.alienvault.com/solutions/security-automation-and-orchestration.

[125] Siemplify, (December 5), *Automation & Orchestration, Security Orchestration introduces order and consistency to your SOC,* [Online]. Available: https://www.siemplify.co/security-orchestration-automation.

[126] MarketsANDMarkets, (January 12), *Security Orchestration Market worth 1682.4 Million USD by 2021,* [Online]. Available: https://www.marketsandmarkets.com/PressReleases/security-orchestration.asp.

[127] B. Kleyman, (October 10, 2014), *Security Orchestration - From data center to cloud,* [Online]. Available: https://blog.algosec.com/2014/04/security-orchestration-data-center-cloud.html.

[128] D. Greenfield, (October 12, 2017), *Should OT Follow IT's Centralized Security Orchestration? ,* [Online]. Available: https://www.automationworld.com/should-ot-follow-its-centralized-security-orchestration.

[129] P. Weeden, (October 12, 2017), *Security Orchestration for Improved Incident Response,* [Online]. Available: https://www.foration.com/blog/security-orchestration-improved-incident-response.

[130] RiskVision, (October 12, 2017), *RiskVision Launches First Out-of-the-Box Security Orchestration Solution with Business, IT and Security Collaboration, Remediation and Analytics,* [Online]. Available: https://www.riskvisioninc.com/riskvision-launches-first-box-security-orchestration-solution-business-security-collaboration-remediation-analytics/.

[131] CyberSponse, (October 13, 2016), *How to Measure the ROI of Security Orchestration and Automation,* [Online]. Available: https://cybersponse.com/how-to-measure-the-roi-of-security-orchestration-and-automation.

[132] S. Bhadra, (October 13, 2015), *Process as code: Security ops orchestration for a brave new world,* [Online]. Available: https://techcrunch.com/2016/03/06/process-as-code-security-ops-orchestration-for-a-brave-new-world/.

[133] Cylance, (January 17), *Security Orchestration and Automation Engineer,* [Online]. Available: https://www.linkedin.com/jobs/view/security-orchestration-and-automation-engineer-at-cylance-inc.-470600981.

[134] L. Musthaler, (October 13, 2013), *Automate security orchestration across platforms, environments,* [Online]. Available:

https://www.networkworld.com/article/2163387/infrastructure-management/automate-security-orchestration-across-platforms-environments.html.
[135] H. N. Security, (October 19, 2016), *Security Orchestration and automation: Closign the gap in incident response,* [Online]. Available: https://www.helpnetsecurity.com/2016/10/07/security-orchestration/.
[136] ForeScout, Automating System - wide Security response through orchestration, White Paper. March 2018, [Online]. Available: https://www.forescout.com/wp-content/uploads/2018/07/FS-WP-Automating_System-Wide_Security-Orchestration_073118.pdf.
[137] R. Howard, The Next Board Problem: Automatic Enterprise Security Orchestration — a Radical Change in Direction, Paloalto Networks, Paloalto Network, Report. Accessed on: August 2017.
[138] M. Wellins, (November 10), *Orchestrating Security Policies - Microsegmentation v Legacy Coonects - Heterogeneous Networks and Hybrid Clouds* [Online]. Available: https://www.tufin.com/resources/videos/video-tufin-orchestrating-security-policies-across-physical-networks-hybrid-cloud.
[139] HEXADITE, (October 31), *What is Security Automation? A guide for an evolving Landscape,* [Online]. Available: http://Hexadite.com.
[140] HEXADITE, (October 23), *Evaluating security orchestration and automation solutions,* [Online]. Available: http://Hexadite.com.
[141] HEXADITE, (October 21), *Security orchestraiton and Automation: Closing the gap in incident response,* [Online]. Available: htttp://Hexadite.com.
[142] Microsoft, (21/01/2018), *Windows Defender Advanced Threat Protection,* [Online]. Available: https://www.microsoft.com/en-us/windowsforbusiness/windows-atp.
[143] FireEye, (November 20, 2017), *Security Orchestration In Action: Integrate – Automate – Manage,* [Online]. Available: https://www2.fireeye.com/Webinar-FSO-EMEA.html?utm_source=fireeye&utm_medium=webinar-page.
[144] ForeScout, ForeScout Agentless Visibility and Control, https://www.forescout.com/wp-content/uploads/2018/08/Agentless-Visibility-and-Control-ForeScout-White-Paper.pdf, White Paper.
[145] ForeScout, Protecting the connection lifecycle - extening visibility, control and orchestration beyond cyber security environments, https://www.forescout.com/wp-content/uploads/2017/04/Protecting-the-Connection-Lifecycle-ForeScout-White-Paper.pdf, White Paper.
[146] ForeScout, ForeScout CounterACT- advanced endpoint visibility for ITAM and CMDB, White Paper. June 2017, [Online]. Available: https://www.forescout.com/wp-content/uploads/2016/12/ForeScout-Advanced-Endpoint-Visibility-for-ITAM-and-CMDB-White-Paper.pdf, Accessed on: August 2017.
[147] Demisto, (October 20), *Collaborative and Automated Security Operations - A comprehensive Incident Management Platform,* [Online]. Available: https://www.demisto.com/.
[148] Demisto, Security Automation and Orchestration - the human perspective.
[149] I. Resilient, (October 23), *Orchestration Platform,* [Online]. Available: https://www.resilientsystems.com/our-platform/ir-orchestration-platform/.
[150] SWIMLANE, (October 20), *Security Orchestration | What is Security Orchestration? ,* [Online]. Available: https://swimlane.com/solutions/security-automation-and-orchestration/security-orchestration/.
[151] A. Stern, (September 24, 2017), *Security Orchestration is more than automation,* [Online]. Available: https://www.siemplify.co/blog/security-orchestration-automation-myth-unmanned-soc.
[152] D. S. Cruzes and T. Dyba, Research synthesis in software engineering: A tertiary study, *Information and Software Technology,* vol. 53, no. 5, pp. 440-455, 2011.
[153] M. B. Miles and A. M. Huberman, *Qualitative data analysis: An expanded sourcebook*. SAGE, 1994.
[154] C. Invotas, (September 21), *Invotas Security Orchestrator,* [Online]. Available: http://invotas.csgi.com/.
[155] C. Seek, (November 20), [Online]. Available: http://cyberseek.org/index.html#about.

[156] R. a. Market, (November 21, 2016), *Security Orchestration Market to Reach $1.6 Billion by 2021 - Rise in Security Breaches & Incidents - Research and Markets,* [Online]. Available: https://www.prnewswire.com/news-releases/security-orchestration-market-to-reach-16-billion-by-2021---rise-in-security-breaches--incidents---research-and-markets-300373845.html.
[157] T. H. Koyama, Kunio; Kitazume, Hideo; Nagafuchi, Mitsuhiro. (2015) Security Orchestration with a Global Threat Intelligence Platform. *NTT Technical Review*.
[158] M. A. Chauhan, M. A. Babar, and Q. Z. Sheng, A Reference Architecture for provisioning of Tools as a Service: Meta-model, Ontologies and Design Elements, *Future Generation Computer Systems,* vol. 69, pp. 41-65, 4// 2017.
[159] C. Islam, M. A. Babar, and S. Nepal, "Automated Interpretation and Integration of Security Tools Using Semantic Knowledge," in *Advanced Information Systems Engineering (CAiSE '19)*, Rome, Italy, 2019.
[160] R. Haesevoets, D. Weyns, and T. Holvoet, Architecture-centric support for adaptive service collaborations, *ACM Trans. Softw. Eng. Methodol.,* vol. 23, no. 1, pp. 1-40, 2014.
[161] IBM, (November 1, 2019), *Orchestrate Incident Response,* [Online]. Available: https://www.ibm.com/security/solutions/orchestrate-incident-response.
[162] J. Andersson and P. Johnson, "Architectural integration styles for large-scale enterprise software systems," in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, Seattle, WA, USA, 2001, pp. 224-236.
[163] T. Miller, Explanation in artificial intelligence: Insights from the social sciences, *Artif. Intell.,* vol. 267, pp. 1-38, 2019.
[164] I. Gorton, *Essential software architecture*. Springer Science & Business Media, 2006.
[165] C. Islam, (2020), *Proof of Concept SOAR,* [Online]. Available: https://github.com/Chadni-Islam/Security-Orchestration-PoC.
[166] Demisto, (January 21, 2020), *Demisto Platform Content Repository,* [Online]. Available: https://github.com/demisto/content.
[167] Siemplify, (December 5, 2019), *What is security orchestration and automation,* [Online]. Available: https://www.siemplify.co/resources/what-is-security-orchestration-automation/.
[168] FireEye, (11/01/2018), *Security Orchestration In Action: Integrate – Automate –Manage,* [Online]. Available: https://www.fireeye.com/solutions/security-orchestrator.html.
[169] J. Oltsik, (25/02/2019, 2017), *ESG Research Report: Cybersecurity Analytics and Operations in Transition,* [Online]. Available: https://www.esg-global.com/research/esg-research-report-cybersecurity-analytics-and-operations-in-transition.
[170] C. Crowley and J. Pescatore, The Definition of SOC-cess? SANS 2018 Security Operations Center Survey, SANS, SANS, 2018, [Online]. Available: https://www.sans.org/reading-room/whitepapers/analyst/membership/38570.
[171] A. Evesti and E. Ovaska, "Ontology-based security adaptation at run-time," in *4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO '10)*, Budapest, Hungary, 2010.
[172] Z. Syed, A. Padia, T. Finin, M. L. Mathews, and A. Joshi, "UCO: A Unified Cybersecurity Ontology," in *AAAI Workshop: Artificial Intelligence for Cyber Security (AAAI, 16)*, 2016.
[173] D. Krauß and C. Thomalla, "Ontology-based detection of cyber-attacks to SCADA-systems in critical infrastructures," in *2016 6th International Conference on Digital Information and Communication Technology and Its Applications, DICTAP 2016*, 2016, Conference Paper.
[174] S. Dua and X. Du, *Data mining and machine learning in cybersecurity*. Auerbach Publications, 2016.
[175] F. Dario, Security orchestration & Automation: parsign the Options, vol. 2017, ed: darkreading, 2017.
[176] Komand, (October 21), *Security automation best practice,* [Online]. Available: https://www.komand.com/.
[177] T. Kenaza and M. Aiash, "Toward an Efficient Ontology-Based Event Correlation in SIEM," in *Procedia Computer Science*, 2016, vol. 83, pp. 139-146.
[178] Demisto, Collaborative and Automated Security Operations - A comprehensive Incident Management Platform, [Online]. Available: https://www.demisto.com/resources/.

[179] A. Evesti and E. Ovaska, "Ontology-based security adaptation at run-time," in *4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2010* 2010: IEEE, pp. 204-212.
[180] E. Casey, G. Back, and S. Barnum, Leveraging CybOX™ to standardize representation and exchange of digital forensic information, *Digital Investigation,* Article vol. 12, no. S1, pp. S102-S110, March 2015.
[181] S. Barnum, Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX), *MITRE Corporation,* vol. 11, pp. 1-22, 2012.
[182] A. Kim, J. Luo, and M. Kang, "Security ontology for annotating resources," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (OTM '05)*, Agia Napa, Cyprus, 2005.
[183] F. Asplund and M. Törngren, The discourse on tool integration beyond technology, a literature survey, *Journal of Systems and Software,* vol. 106, pp. 117-131, 2015.
[184] *Protege,* [Online]. Available: https://protege.stanford.edu/products.php#desktop-protege.
[185] Poveda-Villalón, María, A. Gómez-Pérez, and M. C. Suárez-Figueroa, OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation, *International Journal on Semantic Web and Information Systems (IJSWIS),* vol. 10.2, pp. 7-34, 2014.
[186] M. Bist, A. P. S. Panwar, and V. Kumar, "An agent based architecture using ontology for intrusion detection system," in *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, 2016, pp. 579-587.
[187] A. Razzaq, Z. Anwar, H. F. Ahmad, K. Latif, and F. Munir, Ontology for attack detection: An intelligent approach to web application security, *Computers & Security,* vol. 45, pp. 124-146, September 2014.
[188] C. Islam, M. A. Babar, and S. Nepal, "An Ontology-Driven Approach to Automate the Process of Integration Security Software Systems," in *IEEE/ACM International Conference on Software and System Processes (ICSSP '19)*, Montreal, Canada, 25-26 June, 2019.
[189] Demisto, (January 21, 2020), *Automate the future with Demisto,* [Online]. Available: https://demisto.pan.dev/.
[190] FileExt, (October 5), [Online]. Available: https://filext.com/file-extension/YML.
[191] PaloAlto, (November 20, 2019), *PaloAlto Firewall,* [Online]. Available: https://www.paloaltonetworks.com/.
[192] C. Point, *Check point firewall,* [Online]. Available: https://www.checkpoint.com/products/next-generation-firewall/.
[193] R. Hunddleston, *Introduction to the Grammar of English*. Cambridge University Press, 1984.
[194] L. Rozakis, *The Complete Idiot's Guide to Grammar and Style*. Alpha, 2003.
[195] R. McDonald *et al.*, "Universal dependency annotation for multilingual parsing," in *51st Annual Meeting of the Association for Computational Linguistics (ACL '13)*, Sofia, Bulgaria, 2013.
[196] M.-C. De Marneffe and C. D. Manning, Stanford typed dependencies manual, Technical report, Stanford University, 2008, [Online]. Available: https://nlp.stanford.edu/software/dependencies_manual.pdf.
[197] Spacy, (October 10), *Linguistic Features,* [Online]. Available: https://spacy.io/usage/linguistic-features.
[198] C. Fellbaum, "Theory and applications of ontology: computer applications," 1st ed.: Springer Publishing Company, Incorporated, 2010, pp. 231-243.
[199] G. A. Miller, *WordNet: An electronic lexical database*. MIT press, 1998.
[200] G. A. Miller, WordNet: a lexical database for English, *Communications of the ACM,* vol. 38, no. 11, pp. 39-41, November 1995.
[201] NLTK, (January 17), *WordNet Interface,* [Online]. Available: https://www.nltk.org/howto/wordnet.html.
[202] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, Natural language processing (almost) from scratch, *Journal of machine learning research,* vol. 12, pp. 2493-2537, August 2011.
[203] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "API method recommendation without worrying about the task-API knowledge gap," in *33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*, Montpellier, France, 2018.

[204] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *38th International Conference on Software Engineering (ICSE '16)*, Austin, Texas, 2016.
[205] M. L. McHugh, Interrater reliability: the kappa statistic, *Biochemia medica: Biochemia medica,* vol. 22, no. 3, pp. 276-282, October 2012.
[206] J. R. Landis and G. G. Koch, The measurement of observer agreement for categorical data, *Biometrics,* pp. 159-174, March 1977.
[207] J. Cohen, A coefficient of agreement for nominal scales, *Educational and psychological measurement,* vol. 20, no. 1, pp. 37-46, 1960.
[208] D. M. Jurafsky, James H, *Speech & language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, 2nd ed. Prentice Hall, 2008.
[209] C. D. Manning, C. D. Manning, and H. Schütze, *Foundations of statistical natural language processing*. 55 Hayward St., Cambridge, MA, United States: MIT Press, 1999.
[210] P. University, (2010), *"About WordNet." WordNet,* [Online]. Available: https://wordnet.princeton.edu/.
[211] E. NLP, (November 1), *Deep Dependency,* [Online]. Available: https://emorynlp.github.io/ddr/doc/pages/overview.html.
[212] T. Slimani, Description and evaluation of semantic similarity measures approaches, *International Journal of Computer Applications,* vol. 80, no. 10, pp. 25-33, 2013.
[213] NetworkX, (October 20, 2019), *Software for complex networks,* [Online]. Available: https://networkx.github.io/.
[214] C. Ott, Introducing the Security Orchestration and Automation Playbook: Your Practical Guide to Implementing SOAR, vol. 2019, ed. Rapid 7: Rapid 7, 2019.
[215] SIEMPLIFY, (December 5), *Automation & Orchestration, Security Orchestration introduces order and consistency to your SOC,* [Online]. Available: https://www.siemplify.co/security-orchestration-automation.
[216] D. M. Riehle, S. Jannaber, P. Delfmann, O. Thomas, and J. Becker, "Automatically Annotating Business Process Models with Ontology Concepts at Design-Time," in *International Conference on Conceptual Modeling (ICCM '17)*, 2017.
[217] M. A. Paredes-Valverde, M. Á. Rodríguez-García, A. Ruiz-Martínez, R. Valencia-García, and G. Alor-Hernández, ONLI: An ontology-based system for querying DBpedia using natural language paradigm, *Expert Systems with Applications,* vol. 42, no. 12, pp. 5163-5176, July 2015.
[218] M. Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, and J. J. Samper-Zapater, Ontology-based annotation and retrieval of services in the cloud, *Knowledge-Based Systems,* vol. 56, pp. 15-25, January 2014.
[219] F. Roda and E. Musulin, An ontology-based framework to support intelligent data analysis of sensor measurements, *Expert Systems with Applications,* vol. 41, no. 17, pp. 7914-7926, December 2014.
[220] M. Reformat and C. Ly, Ontological approach to development of computing with words based systems, *International Journal of Approximate Reasoning,* vol. 50, no. 1, pp. 72-91, January 2009.
[221] R. Y. Lau, C. C. Lai, J. Ma, and Y. Li, "Automatic domain ontology extraction for context-sensitive opinion mining," in *International Conference on Information Systems (ICIS '09)*, Phoenix, Arizona, USA, 2009.
[222] T. D. Nguyen, A. T. Nguyen, H. D. Phan, and T. N. Nguyen, "Exploring API embedding for API usages and applications," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE '17)*, Buenos Aires, Argentina, 2017.
[223] A. T. Nguyen, P. C. Rigby, T. V. Nguyen, M. Karanfil, and T. N. Nguyen, "Statistical Translation of English Texts to API Code Templates," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, Buenos Aires, Argentina, 20-28 May 2017, 2017.
[224] M. M. Rahman, C. K. Roy, and D. Lo, "Rack: Automatic api recommendation using crowdsourced knowledge," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER '16)*, Suita, Japan, 2016.

[225] R. B. Bohn, J. Messina, F. Liu, J. Tong, and J. Mao, "NIST cloud computing reference architecture," in *2011 IEEE World Congress on Services*, 2011: IEEE, pp. 594-596.
[226] O. C. A. W. Group, OpenFog reference architecture for fog computing, *OPFRA001,* vol. 20817, p. 162, 2017.
[227] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (SDN): A reference architecture and open APIs," in *2012 International Conference on ICT Convergence (ICTC)*, 2012: IEEE, pp. 360-361.
[228] N. K. Tran, "A Reference Architecture and a Software Platform for Engineering Internet of Things Search Engines," 2018.