Architecture-Centric Support for Integrating Security Tools in a Security Orchestration Platform

Chadni Islam[1,2](✉), Muhammad Ali Babar[1], and Surya Nepal[2]

[1] CREST Centre, University of Adelaide, Adelaide, SA 5005, Australia
{chadni.islam,ali.babar}@adelaide.edu.au
[2] CSIRO's Data61, Sydney, NSW, Australia
{chadni.islam,surya.nepal}@data61.csiro.au

Abstract. Security Operation Centers (SOC) leverage a number of tools to detect, thwart and deal with security attacks. One of the key challenges of SOC is to quickly integrate security tools and operational activities. To address this challenge, an increasing number of organizations are using Security Orchestration, Automation and Response (SOAR) platforms, whose design needs suitable architectural support. This paper presents our work on architecture-centric support for designing a SOAR platform. Our approach consists of a conceptual map of SOAR platform and the key dimensions of an architecture design space. We have demonstrated the use of the approach in designing and implementing a Proof of Concept (PoC) SOAR platform for (i) automated integration of security tools and (ii) automated interpretation of activities to execute incident response processes. We also report a preliminary evaluation of the proposed architectural support for improving a SOAR's design.

Keywords: Security orchestration · Security automation · Software architecture · Security tool integration · Design space

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 team [1–4]. SOAR platforms enable integration, orchestration and automation of the activities (e.g., block IP, scan endpoint and isolate host) performed by security tools and human experts [2].

Existing SOAR platforms lack proper abstractions for designing a platform at the architectural level [1–3, 5]. Most of the existing SOAR platforms are implemented in ad-hoc manners without much attention to the underlying infrastructure [2]. As a result, there can be several engineering challenges involved in embedding agility in a SOAR platform [2, 4, 7]. These challenges result in highly complex and monolithic design that is hard to evolve overtime. A SOAR' design complexity may also worsened by a lack of conceptual and practical guidelines for optimal architectural design decisions [2, 6].

An architecture-centric approach [7–9] 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 [7], which can be improved by leveraging suitable architectural styles and patterns.

A design space is required to capture and characterize design decisions for integrating techniques and tools that underpin a SOAR platform [2]. Developing design spaces for different domains of software systems is a growing trend [7]. The design space of a SOAR platform involves many architectural design decisions and trade-offs that are impacted by security tools and applications integrated into these platforms. We propose a concept map considering the functionalities performed by a SOAR platform. It allows one to modularize the functions and separate the concerns of the components that provide the design space of a SOAR platform.

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

– *Abstraction to model 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 requirements of emerging SOAR platforms and integration technologies for faster response and efficiency.
– *Layered Architecture for SOAR platform*: 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 mean for delimiting the design space.
– *Proof of concept SOAR support*: We have developed a Proof of Concept (PoC) SOAR platform that has been designed to fulfill the quality requirements - *integrability*, *interpretability* and *interoperability* 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 paper is organized as follows. Section 2 introduces a concept map of a SOAR platforms' design space. Section 3 presents the modularized architecture of a SOAR platform. Section 4 details the dimension of a SOAR platform's integration design space. Section 5 presents a case study. Section 6 demonstrates the evaluation of the PoC. Section 7 discusses related work and Sect. 8 concludes the paper.

2 Security Orchestration and Automation

The SOAR platforms are integrated solutions for an organization's SOC. The underlying technologies of SOAR platforms are designed to interweave people, process and technology. In a SOAR platform, people are responsible for intelligence-based decision making and technologies are used to streamline complex process. 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 [2].

The development of any SOAR platform first needs to focus on *integrating* the security tools in a single platform. Depending on the organizations, the security tools can be open source, commercial, proprietary, packaged or even legacy bunch of scripts. Security tools are generally integrated using plugins, scripts, APIs and modules. Mostly SOAR vendors provide plugins and APIs based support for 150–200 security tools [10, 11]. Security tools generate data in a variety of formats. Further, 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 makes security tools interoperate in a manner where 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.

2.1 Functional Requirements of Security Orchestration and Automation

SOAR as a Unifier or Hub. We adopt the functionality of a SOAR outlined in a recent multivocal review [2]. We consider a SOAR platform as a hub that unifies the activities of security tools and provides a single pane for supporting 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 [12]. 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 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 semantics correctness before delivering them to analytics tools. A SOAR's architecture should support semantics integration among the artifacts produced and consumed by security tools.

SOAR as a Coordinator or Orchestrator. A SOAR platform orchestrates 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 a lot of SOAR literatures tend to use integration mechanisms or connecting tools as an umbrella term to cover all processes that happen under-the-hood 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.

2.2 Quality Attribute Requirements

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

- **Integrability**: 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 modification of security tools' functionalities.
- **Interoperability**: 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.
- **Interpretability**: A SOAR platform should be able to semantically interpret the data generated and consumed by security tools.
- **Flexibility**: A SOAR platform's tasks depend on IRPs and emerging threat behavior which changes continuously. A SOAR architecture should be flexible to provide mapping support for security tools and IRPs to adapt the changes.
- **Usability**: 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.

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 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 (Fig. 1). This categorization

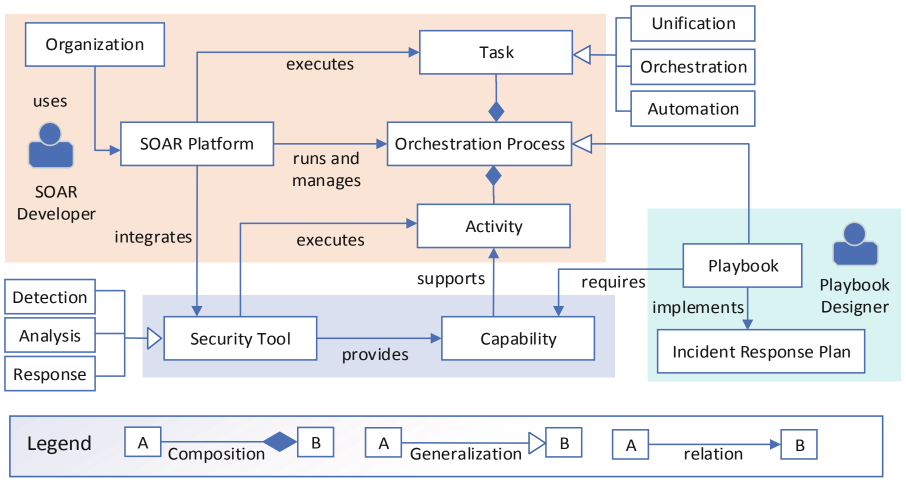

Fig. 1. Conceptual map of security orchestration and automation

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 security tools used for this research is out of the scope of this paper.

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 (Fig. 1). A SOAR platform performs a set of tasks that can be categorized in 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 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 task to respond to a security incident.

The orchestration process primarily is 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 Team (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 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 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 tool, (ii) integration, (iii) data processing, (iv) semantic, (v) orchestration and (vi) User Interface (UI) layer as shown in Fig. 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 staff to freely choose the preferred components and deploy a SOAR based on their requirements (Fig. 2).

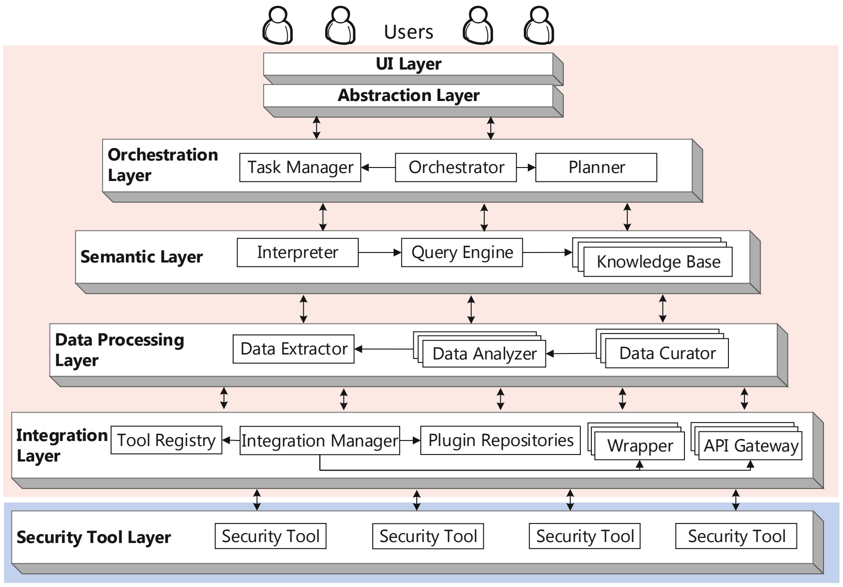

Fig. 2. High-level architecture for SOAR platform

Each layer is decomposed into components and sub-components. We consider the components as the lower level of abstractions. Figure 2 shows the core components and interactions among the components that are required to achieve the desire 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.

UI Layer: Security staff initiate existing IRPs or define new plans using a SOAR's User Interfaces (UIs) such as interactive dashboards or Integrated Development Environment (IDE) or Command Line Interface (CLI). The UI layer supports *flexibility* in designing UIs that helps 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 can be implemented as part of the UI layers to maintain and encapsulate the interaction among a SOAR's user and its components (Fig. 2).

Orchestration Layer: The *orchestrator* and *task manager* together form the coordinator of a SOAR platform (Fig. 2). The orchestrator is responsible for coordinating and forming configuration to achieve *interoperability* and automating 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 contains the details of the process about the input required to execute a task and also the output that is generated after task execution. The playbooks further contain the conditions that trigger the execution of a particular 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.

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*. A knowledge base usually consists of an *ontology* of security tools, their capabilities and activities of an IRP, which enables the *interpreter* to semantically interpret security tools' capability and IRPs' activities. The details of the ontology can be found in [13]. The *query engine* is responsible for extracting data from a knowledge base. In our proposed architecture, we consider the semantic layer separate from other layers to give SOC the *flexibility* to define or modify an ontology.

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. *Data curator*, *data extractor* and *data analyzer* are the three main components of 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 throughout an IRP execution process. An architect can incorporate any automation algorithm or data analysis techniques as part of data analyzer without affecting other components of a SOAR.

Integration Layer: The integration layer has five components: *integration manager*, *wrapper*, *tool registry*, *plugin repository* and *API gateway*. This layer is designed to integrate security tools. The *integration manager* works as a description module through which security tools are integrated and information is provided to enable *interpretability* 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 orchestrator's commands. The difference between wrapper, plugins and API gateway lies in security tools integration and communication protocols.

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-of-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 are necessary to integrate them into a SOAR platform.

Figure 3 shows an example UML sequence diagram for responding to a security incident that comprises of components from each layer.

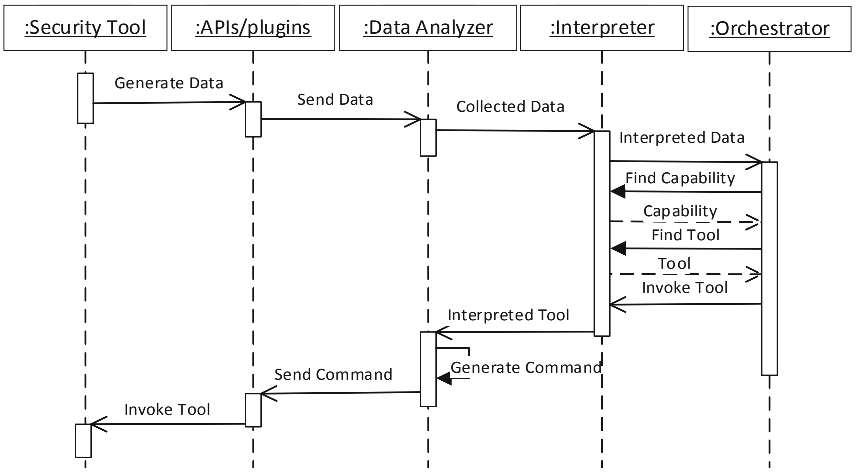

Fig. 3. An example sequence diagram showing the flow of data and interaction of components

4 Dimensions of the Design Space of SOAR

The design space of a SOAR 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.

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 *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 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 IRPs activities. Here, the modular architecture helps with defining different processes, which are mainly orchestration of security tools, SOAR's components and organizational information systems.

A SOAR platform can be centralized, distributed or hybrid depending on an organization's infrastructure [2]. 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.

Technology Decision: 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. A SOAR's taxonomy has six automation strategies [2]. An underlying technology infrastructure consists of the assets of an organization depending on the type of the automation strategy. Example of assets includes 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 [12].

Following 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 IRP and where to deploy them in an organization's environment so that 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 discovery of 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.

Table 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 system wide impact and/or impact on more than one non-functional requirements [8].

Table 1. Summary of architectural design decision

Design decisions	Expected benefits
Ontology for formalizing security tools and activities of IRPs	Make a SOAR architecture flexible 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 dynamic environment
Layered architectural style	Easy evolution of SOAR's components and easy modularization of functionalities and components
Abstraction of SOAR's components task with a set of APIs	Make a SOAR platform easy to use, manage and learn for end-users
Automated integration and interpretation process	Enable reuse of existing components with changes in IRPs and security tools
Share ontology template in a centralized repository pattern	Provide access of the ontology to its end users and support flexibility in update

5 Case Study – Prototype Implementation

In this section, we present a Proof of Concept (PoC) SOAR platform that we have designed and implemented based on the proposed architectural approach [14]. The functional requirements of our PoC 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 the PoC in a way so that it is easily evolvable for future changes. In this implementation, we considered two types of changes that are most common is SOARs execution environment – change in security tools and change in IRPs. Figure 4 presents the implementation architecture of the PoC. We analyzed the instruction of 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 selected seven open-source tools[1] 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 Tool), EDR (Endpoint Detection and Response) tool, Open Source Threat Intelligence and Sharing Platform (OSINT), Firewall and packet monitoring and logging tools respectively. The security tools were selected based on the diversity in their capabilities because execution of an IRP would require multiple security tools. 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)

[1] https://www.snort.org, https://www.splunk.com/, https://www.limacharlie.io/, https://www.misp-project.org.

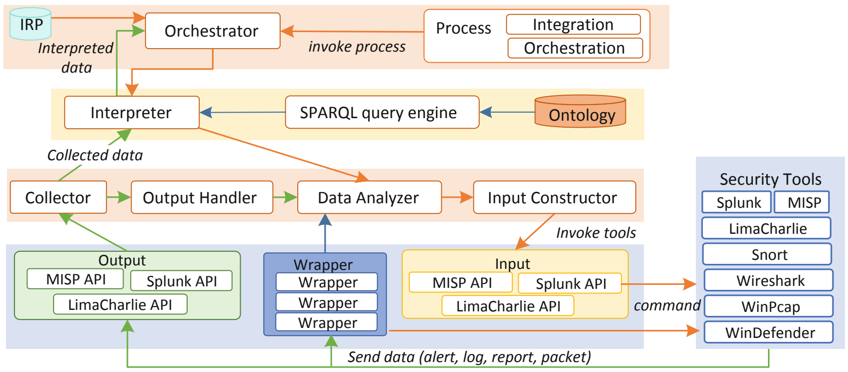

Fig. 4. Implementation architecture of the PoC for security tool integration

collaborative playbooks [15]. 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 require to execute without user intervention. The list of capabilities and IRPs are available at [14].

The implementation decision incorporated APIs based integration style as our primary mechanism to integrate security tools into a SOAR. The data from the security tools such as MISP and Splunk have been made accessible through their APIs. Besides, we have built wrappers for security tools that do not provide specific APIs such as Snort. Integrating a new tool required us to identify security tool's APIs or information sharing protocol and implement a suitable integration mechanism. The API and wrappers of Fig. 4 are part of the *integration layer* of the PoC.

We also designed an ontology to formalize security tools, their capabilities and IRP's activities to enable semantic interpretation of security tools data [13]. Each security tool can execute multiple activities and each activity can be executed by multiple security tools. We used Apache Jena Fuseki server to store the ontology. Security tools are formalized based on their capabilities and the activities of IRPs are mapped with security tool class of an ontology. Table 2 and Table 3 illustrate how 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*.

We built a collector to gather security tools' data, which are sent to an orchestrator via the interpreter for actions, e.g., Splunks 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. Further to formulate the commands, an input constructor is built.

The automation algorithms or processes have been mainly built as integration processes that are the parts of the orchestration layer (Fig. 4). We designed and implemented scripts to define the automated integration process, which includes selecting the security tools based on activity description, interpreting their capabilities, formulating the input commands and finally invoking the security tool by calling appropriate APIs [16]. An example is shown in Fig. 5 where the output of Splunk is sent to LimaCharlie. The

Table 2. Illustration of a selected set of object properties of security tool class of an ontology

Security tool	Security tool class	has Capability	Capability class	executeActivity
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. Illustration of a selected set of data properties of security tool class of an ontology

Security tool	Security tool class	isIntegrated	hasInputType	hasRule	hasConfigFile
snort_s	IDS	True	Network traffic	False	snorts.config
limaCharlie_l	EDR	True	Payloads	True	inputs.conf
splunk_s	SIEM	True	Logs	True	LCConf

orchestrator is required to collect the output of Splunk and then interpret it. All the data generated by Splunk might not be required by LimaCharlie; so, the orchestrator would require to construct the input of LimaCharlie from Splunk's output to invoke LimaCharlie. We developed and designed this process as part of the integration process to automate the interpretation of the security tools data, which enable seamless interoperability among security tools. Using the integration process, data sharing among the security tools of Fig. 5 happened seamlessly.

6 Evaluation

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

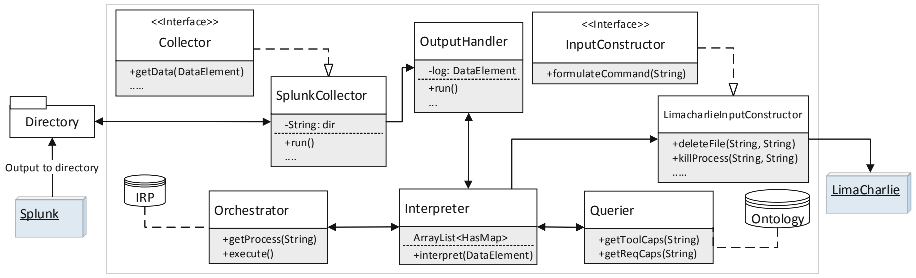

Fig. 5. Example of data transfer from Splunk to LimaCharlie

6.1 Automating the Process for Integration Security Tools

Let's assume that a user has expressed a goal of integrating security tools. 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 for specifying the tools' capabilities and map it with the available activities or activity it can execute. 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. Further, the process for automating the integration of security tools is invoked which enables the collector to collect the security tools' output and 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 to develop 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 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 while considering API and wrapper-based integration for taking the output and provide the commands to execute an activity. An 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 require 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, addition of new capabilities or change in the existing API parameters also requires designing the connections and updating the playbook where the security tools have been used.

With the semantic-based integration approach, we only need to update security tools details in an ontology. The connections between the ontology and other components have already been designed and that do not require any changes. Thus, with the PoC, the number of components and connections remain constant with the integration of new security tools – that is MISP. Without considering the proposed architecture approach the number of components increased at least by 2 upon the 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 components and connections lower by reusing the existing components.

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

6.2 Automating the Interpretation of the Activities to Execute an IRP

We assume a user has expressed his/her goal 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 require to manually identify the security tools. He/she just needs to request the orchestrator for 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 would 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 a 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 generated data of 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 requires 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 in adapting the changes and the evolution of a system does not increase the complexity of architecture. Further, it has reduced the overhead for users in adopting the changes by providing the 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 used security tools 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 compared to other approaches.

This paper has demonstrated the feasibility of the proposed architecture for security tool integration and IRP interpretation based on three quality attributes - *integrability*, *interpretability* and *interoperability*. Other quality attributes of a SOAR can be evaluated by following different architectural evaluation techniques such as Scenario based Architecture Analysis Method (SAAM) and Architecture Tradeoff Analysis Method (ATAM) [8, 17].

7 Related Work

The leading security service providers aim to provide SOAR platforms to deliver end to end security services [10, 11, 18, 19]. For example, FireEye (i.e., a leading cybersecurity company) designs a SOAR platform to integrate its endpoint products and offer supports to its industry partners [10]. Whilst the start-ups mainly focus on developing APIs to integrate different third-party solutions and provide playbooks for automated and semi-automate IRPs [20]. 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. Further, a SOAR is a large-scale system that integrates an organization's information and security systems. Organizations are facing 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 [2, 13]. 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 [1, 3, 4, 21, 22]. 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 in IRPs execution. In these studies, a SOAR platform has mainly focused on security tools 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 platforms 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 [1, 3, 4, 22].

None of the existing works 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 [6–8]. An increased focus on architectural aspects of SOAR can also facilitate further research on the design decisions of the exiting 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 organization 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' efficiency.

8 Conclusion

Exploring and understanding the architectural design decision before designing and implementing a SOAR platform is a valuable task. The captured design decision would help developers as well as a SOC staff of an organization to systemize their decision process and trade-off analysis. The architectural design decisions would serve as a standalone lexicon to describe and evaluate the existing and new SOAR platform. In this paper, we have designed a conceptual diagram of 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 the 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 the consideration of the principal dimension of the architecture design space has improved SOAR design practices.

The proposed approach has further laid a foundation for future research on the design space and deployment automation of SOAR platforms. In our future work, we plan to conduct a large-scale mapping of the existing SOAR platform and IRPs onto the architecture design decisions to generate patterns and hide interaction among the different components across multiple technology paradigm.

Acknowledgement. This work is partially supported by CSIRO's data61, Australia. We acknowledge the contribution of Faheem Ullah, Aufeef Chauhan and Triet Mihn Le for their feedbacks in improving the work.

References

1. Feitosa, E., Souto, E., Sadok, D.H.: An orchestration approach for unwanted internet traffic identification. Comput. Netw. **56**(12), 2805–2831 (2012)
2. Islam, C., Babar, M.A., Nepal, S.: A multi-vocal review of security orchestration. ACM Comput. Surv. (CSUR) **52**(2), 37 (2019)
3. Luo, S., Salem, M.B.: Orchestration of software-defined security services. In: 2016 IEEE International Conference on Communications Workshops (ICC 2016), Kuala Lumpur, Malaysia (2016)
4. Nadkarni, H.: Security orchestration framework. US Patent 9,807,118 (2017)
5. Koyama, T., Hu, B., Nagafuchi, Y., Shioji, E., Takahashi, K.: Security orchestration with a global threat intelligence platform. NTT Tech. Rev. **13**, 1–6 (2015)
6. Chauhan, M.A., Babar, M.A., Sheng, Q.Z.: A reference architecture for provisioning of tools as a service: meta-model, ontologies and design elements. Future Gener. Comput. Syst. **69**, 41–65 (2017)
7. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, USA (2005)
8. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Professional, Boston (2003)
9. Haesevoets, R., Weyns, D., Holvoet, T.: Architecture-centric support for adaptive service collaborations. ACM Trans. Softw. Eng. Methodol. **23**(1), 1–40 (2014)
10. FireEye.: Security orchestration in action: integrate – automate –manage. https://www2.fireeye.com/Webinar-FSO-EMEA.html?utm_source=fireeye&utm_medium=webinar-page. Accessed 20 Nov 2017

11. IBM.: Orchestrate incident response. https://www.ibm.com/security/solutions/orchestrate-incident-response. Accessed 1 Nov 2019
12. Andersson, J., Johnson, P.: Architectural integration styles for large-scale enterprise software systems. In: Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference, Seattle, WA, USA, pp. 224–236 (2001)
13. Islam, C., Babar, M.A., Nepal, S.: Automated interpretation and integration of security tools using semantic knowledge. In: Advanced Information Systems Engineering (CAiSE 2019), Rome, Italy (2019)
14. Islam, C.: Proof of concept SOAR (2020). https://github.com/Chadni-Islam/Security-Orchestration-PoC
15. Demisto.: Demisto platform content repository. https://github.com/demisto/content. Accessed 21 Jan 2020
16. Islam, C., Babar, M.A., Nepal, S.: An ontology-driven approach to automate the process of integration security software systems. In: IEEE/ACM International Conference on Software and System Processes (ICSSP 2019), Montreal, Canada, 25–26 June (2019)
17. Babar, M.A., Zhu, L., Jeffery, R.: A framework for classifying and comparing software architecture evaluation methods. In: Proceedings of 2004 Australian Software Engineering Conference, pp. 309–318 (2004)
18. Siemplify.: What is security orchestration and automation?. https://www.siemplify.co/resources/what-is-security-orchestration-automation/. Accessed 5 Dec 2019
19. Swimlane.: Security automation and orchestration. https://swimlane.com/use-cases/security-orchestration-for-automated-defense/. Accessed 20 Nov 2017
20. Demisto.: Security orchestration and automation. https://www.demisto.com/wp-content/uploads/2017/04/MH-Demisto-Security-Automation-WP.pdf. Accessed 5 Dec 2017
21. Digiambattista, E.: Enterprise level security orchestration. US Patent 2017/0017795 A1 (2017)
22. Poornachandran, R., Shahidzadeh, S., Das, S., Zimmer, V.J., Vashisth, S., Sharma, P.: Premises-aware security and policy orchestration. US Patent 14/560,141 (2016)