A Hierarchical Rule-based Security Management System for Data-Intensive Applications

Yar Rouf

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGIES

YORK UNIVERSITY

TORONTO, ONTARIO

SEPTEMBER 2018

Abstract

Applications in today's software development environment evolve at a rapid rate, constantly providing their users with new functionalities. As a result, it becomes increasingly complex to understand the entire application. The security team and the developers may not completely understand each other's approaches, resulting in a less secure system with vulnerabilities. In addition, there is large amount of security data to be analyzed. To mitigate these issues, we propose a platform to support the SecDevOps framework, a hierarchical distributed architecture for security control that uses a Business Rules Engine (BRE). The BRE simplifies security rules by allowing the teams to write them at an operational level rather than at the network level, which requires specialized knowledge. Business rules are universally understood by the different teams, resulting in effective inter-team communication. Additionally, the platform can expand and scale with new security rules and data sources at runtime in a systematic manner.

Acknowledgement

Firstly, I would like to express my gratitude to my supervisor, Professor Marin Litoiu. Under his guidance, I was able to get involved in fields of Cloud Computing and Autonomic Computing, and subsequently this research accomplishment. Professor Litoiu would constantly provide new perspectives and insights to my research that were invaluable to my progress in the Master's program. My opportunity under Professor Marin Litoiu was filled with new experiences, lessons, support and encouragement. Without Professor Marin Litoiu, this thesis would not have been possible.

I am genuinely appreciative of the assistance to Dr. Mark Shtern and Dr. Marios Fokaefs. With their combined knowledge of Computer Security and Software Engineering, their vision and expertise were essential for this accomplishment. I am deeply grateful of the valuable research and technical experience that I gained working under them. Their support and involvement was vital to the thesis. In addition, I kindly thank Dr. Joydeep Mukherjee for generously providing a large amount of positive feedback and constructive criticism to the text and structure of the thesis.

I give my thanks to the committee members Dr. Zijiang Yang and Dr. Natalija Vlajic for providing valuable feedback and interesting viewpoints on my research during the thesis defence.

I would also like to express my gratitude to all the members I worked with at CERAS labs and Professor Marin Litoiu research team. Dr. Hamzeh Khazaei, Dr. Cornel Barna, Justin Li, Paul Vytas, Adam Di Prospero, Brian Rampasad, Calin Armenean, Ali Zargar Shabestari, Nasim Beigi Mohammadi and Amar Patel. Thanks for allowing me to work with and alongside all of you, and sharing your knowledge.

Finally, I want to dedicate this work to my father Niaz, my mother Shahina, my brother Raphael and many of my friends who supporting me throughout my studies as a Graduate student. All of your presences have brought immeasurable positive energy in my life. Thank you for all the unconditional love and constant support.

Table of Contents

List of Figures

Chapter 1

Introduction

Software development is fast-paced and constantly growing in terms of complexity and size when adding new functionalities [1]. Throughout this process, the development, operations and security teams are responsible to comply with business and security requirements of the organization [2]. These include effective protection and mitigation against malicious attacks, suspicious traffic, application bugs and exploits.

1.1 Problem and Motivation

Vulnerabilities that arise in software development can be a serious threat to businesses, leading to security breaches. These breaches are still a major problem today. As shown by the 2017 Ponemon Security study [3], sponsored by IBM, the average global cost of a data breach is \$3.62 million. The average size of a data breach has increased by 1.8% in a sample of 439 companies. According to the study, the data breaches ranged from 2,600 to slightly less than 100,000 compromised records, thus showing that data breaches are still a significant and valid threat. Furthermore, 47% of these data breaches are from malicious or criminal attacks, and 25% of these breaches are caused by a system glitch.

Vulnerabilities in web applications, which expose the application to attacks, are due to the various bugs and violations from the software. The software development team follows a well-defined process to fix the bugs that may take from several days to months to apply the patches depending on the seriousness of the bug [4]. During this time frame, a large data breach can occur due to software error and the aftermath is the loss of jobs, compromised data, reputation loss, and diminished goodwill [5]. This potential outcome weighs pressure on the security team, which leads them to find solutions and preventative measures against software error without delay. Software developers are in charge of building the application functionalities in regards to the requirements, which requires extensive knowledge of programming languages, libraries and other development tools. Security analysts, on the other hand, focus on protecting the application from malicious attacks and breaches. The developer and the security teams use different terminologies and tools to describe their individual responsibilities. Seldom will the security analyst fully understand the fine details of the source code and, similarly, the developers will not have complete knowledge of the various security tools and languages such as policies and alerts. This leads to a barrier in communication between developers and security analysts, even within the limited scope of a small scaled enterprise.

It is becoming increasingly clear that effective communication is important for a stronger, secure system. DevOps [6] has emerged as a new popular practice, where the development and operations teams collaborate in a more unified process. This practice increases the efficiency of software development and it produces higher-quality software. Incorporating security in this process has coined a new term, *SecDevOps*, involving the collaboration between the development team, the operation team and the security team [7]. In SecDevOps, the most important stages where security concerns are incorporated are static code analysis, dynamic code analysis, and runtime application protection [8]. In static code analysis, the security team along with the development team will review the code of the application during development or when it is pushed into the code repository [8]. Dynamic code analysis is scanning the web application while it is running, usually during Quality Assurance (QA), to detect for vulnerabilities that may not appear or be difficult to identify by simply reviewing the code [8]. Runtime application protection is the act of monitoring and protecting the web application while it is in the production environment, from malicious users and suspicious traffic [8]. In this research, our tool focuses on supporting the latter two stages, where we monitor the application during QA and production.

To help support SecDevOps and bridge the communication gap between the teams, we propose an between the software development and security teams. The intermediate language is supported by a stateful Business Rule Engine (BRE), a software component that interprets the set of business rules [9] and runs these rules against the specified business dataset. The Business Rules Group defined business rules in a information system context as a statement that defines or constrains some aspect of the business by asserting business structure, controlling or influencing the behavior of the business [10]. The business rules are essentially the requirements of the organization, in our case, the security requirements of the software system. The dataset are all the objects we wish to assert the specified constraints. In our research, the dataset includes the collection of logs, database data, sensor data and any information that can help us apply security measures to the software system. The BRE creates events after a rule is triggered based on monitored data. The stateful property of the BRE allows the events stored to be used by complex rules, and activate actions based on a rule. This makes the BRE a powerful tool to express security and development concerns and facilitates communication based on these rules.

Although BREs are useful for effective communication in an enterprise, they have their own issues, namely the scalability issue. Since the ruleset has the potential to become extremely large due to the requirements of the security and development teams, the BRE by itself will begin to have performance issues and not respond fast enough to security alerts. The growing number of objects, the number of rules, the number of concurrent callers in the BRE will impact the performance of rule execution [9]. Another problem that arises is that a large ruleset can be complex to interpret and maintain. When a complex security requirement or policy has been updated, reviewing one single large ruleset with thousands of security rules can become time-consuming, confusing and prone to incomplete rule modification. This results in a maintainability problem.

When monitoring a large-scale system, there are multiple data sources that can be analyzed to check for security violations, attacks and suspicious behavior. These can include access logs, audit logs, sensor data and others. This can result in a large amount of rules of different data types, which may become a performance problem as stated earlier. In addition, the rules should be able to easily correlate the incoming data and events (e.g. Brute Force attack or Cross-Site Scripting Attack) to detect more sophisticated attacks. We can divide the rulesets into smaller rulesets in a hierarchical structure. Each ruleset is responsible for a certain data type or a specific attack, and events generated by multiple rulesets can be sent to the next ruleset in the hierarchy for further analysis. The events that are generated are the security threats and additional information surrounding it, for e.g., Brute Force attack with IP address of attacker, username, number of attempts, etc. This ruleset can aggregate the events and correlate them to detect a complex and system-wide attack.

To support SecDevOps and satisfy the goals of scalability, maintainability and flexibility, we implement the rule-based engine with a scalable and hierarchical Streaming Analytics Clusters, enabling us to create a flexible architecture enabling SecDevOps and enforcing a more secure system. Organizing the Analytics cluster in a hierarchy allows us to divide a large ruleset in smaller rulesets, to mitigate issues of maintainability and performance. In addition, when there is an increase in demand, the Analytics cluster also gives us the advantage of adding or removing the nodes that perform the analysis. This allows us to employ our BRE on both QA and In-Production applications.

1.2 Research Objectives and Questions

The main objective of this thesis is to incorporate a BRE into an analytics streaming cluster to simplify security control and to be used as a platform to support the SecDevOps approach. To achieve this objective, it is important to address these following research questions (RQs):

- **RQ-1**: How can we simplify security control into a set of business rules allowing Developers and Security Analysts to communicate more effectively?
- **RQ-2**: How can we use the Business Rules Engine and Analytics architecture to aggregate data from several sources to understand the complete state of an application cloud environment?
- **RQ-3**: How can we quickly respond and mitigate security threats against the system?

- **RQ-4**: Is a Rules engine an effective solution to protect against application-specific vulnerabilities?

The RQs are described in details below:

RQ-1: Business rules have been used mainly for business policies and rules of the specified industry. These can be checks for financial services, such as mortgage underwriting, tax reporting, and insurance services, such as policy underwriting and claim processing [9]. Business rules are easy to understand allows non-programmers to read and understand business rules for financial and insurance related uses. However, in this research, we have to design and write business rules for a more technological-focused field, namely software engineering and computer security. Unlike datasets of policy claims and tax reports, the incoming security and software-related data is generated continuously at large volume, and can be widely varied and might contain noise. In our research, we must be able to organize the incoming security data and alleviate the complexity that may arise from such large volumes of incoming data.

RQ-2: A large-scale system has many components and sources of data, such as logs or security sensors. With a large amount of users today, large-scale systems are distributed throughout a network. As a result, events such as security attacks that can be easily detected in one component can be hard to detect simultaneously across several components belonging to the system. A single fault detection system or sensor is insufficient to handle events for a large-scale distributed system. Such a detection system will eventually cause information loss at high volumes of incoming data since the system will drop packets at high load [11]. Alternatively, distributing sensors will only give us the partial information of the system. Hence, our BRE and Analytics architecture should be able to aggregate and analyze large volumes of data from multiple system components in order to correlate information for understanding the complete picture in a system-wide context.

RQ-3: While our solution monitors and analyzes the software systems logs for security threats, there needs to be a way for security analysts to respond and mitigate attacks. Traditionally, after the collection and analysis process, security analysts will begin planning a preventative strategy to secure the system. What we are interested in is using the BRE to execute an autonomous action to mitigate the security threat. This allows for security threats to be blocked permanently or temporarily from further tampering the system while the security analysts can work on a patch, if needed, against this security threat.

RQ-4: An issue that is highly relevant in computer security is an effective way to detect application-specific vulnerabilities. Distributed Denial of Service (DDoS) attacks, Brute Force attacks, malware detection are generic attacks that can be detected. However, an application-specific exploit originates due to unknown software bugs or coding issues. These application-specific vulnerabilities are not a generic type of attack that can be easily detected, and can only be found with clear knowledge of the application. In our research, we want to see if directly involving the software developers with the security teams to write security rules can help reduce and mitigate application-specific vulnerabilities.

1.3 Thesis Contributions

The work developed in this thesis contributes to computer security control and management of a large-scale system or application. Our research objective is to support the practices of SecDevOps, which will help simplify the process to more secure applications. The research contributions of this thesis are:

- We first design an architecture that employs a Streaming Analytics Cluster that uses a BRE that analyzes incoming data (i.e. logs, sensors and database) of the system against customizable business rules. The business rules are the security requirements of the organization and the BRE provides a more simplified way of defining security rules at the operational level, allowing developers and security analysts to communicate more effectively. This architecture follows the principles of Cloud Computing, distributing workers and streaming clusters throughout the network and allowing for several Streaming Analytics Clusters for distributing a large ruleset. This architecture distributes the cluster in a hierarchy, with each cluster responsible for a set of business rules. Incoming data is then streamed into the architecture and ran against the ruleset. When a rule triggers, events are created and an adaptive action can be executed. The events can also be streamed into the next cluster in the hierarchy for more sophisticated analysis and actions. The clusters can be scaled in response to performance and load by increasing or decreasing the number of nodes in the cluster. This architecture and the BRE as a platform for the developers and security analysts are the baseline work of this thesis. This contribution addresses the first two research questions, as we provide a platform that can simplify security control and aggregate multiple data sources.

- We design several adaptive functions for the BRE to mitigate attacks. This will be the Execution portion of the MAPE-K (Monitor, Analyze, Planning, Execution) Loop, which is the process to make the architecture autonomous. With the addition of adaptive functions, we have completed the MAPE-K Loop with our architecture. For our architecture, we created functions to block IP addresses, alerting users and admins of the application and removing documents from the database. These functions help us mitigate the attacks from our use case and case study which are explained in the next two thesis contributions. This contribution addresses the third research question, where we use these adaptive actions to respond and mitigate the security threats.

- We implement the architecture with various practical security use cases that occur in a real world setting. We use the designed architecture to demonstrate how it can be used as a solution to these threats and issues that are present in today's computing era. These use cases show the architecture's ability to be domain efficient and easily integrate with different applications and attacks.

  The security use cases we implemented our architecture with were DDoS detection, compromised accounts, data breaches, library vulnerabilities, software evolution and application-specific vulnerabilities. For DDoS attacks, we were able to distribute the sensors and aggregate the incoming sensor data with our architecture. As a result, we are able to detect and mitigate a DDoS attack by understanding the complete state of our system and without any information loss due to a single sensor. We protect against compromised accounts by using our BRE to correlate Brute Force attacks with suspicious logins. Another serious security issue are data breaches, which significantly affects many organizations. We successfully use our BRE to create security policies for a database, such as threshold rules, and when these rules are violated, we detect suspicious access of the database.

  The library vulnerability use case is designed to showcase our architecture's ability to correlate 3rd party data with application data. To speed up development, developers would use many libraries to create their applications. However, libraries may have vulnerabilities, and these vulnerabilities are stored in a Common Vulnerabilities and Exposures (CVE) list. We successfully stream the CVE list into the architecture, and due to our architecture's ability to be stateful, this CVE list is in our engine's memory. When developers introduce libraries, we stream that into our architecture and match it against the CVE stream to find any vulnerabilities. In the Software Evolution use case, when new functionalities are added to an application, our architecture allows us to write rules at run-time to make sure these functionalities are working as intended. The application-specific use case showcases our architecture's ability to detect and mitigate against vulnerabilities and exploits, due to poor code and bugs of a web application. This contribution addresses all the research questions through implementing our platform with each of the platforms. For the application-specific use case addresses the fourth research question specifically.

- We apply the architecture on a data-intensive industrial application to examine its effectiveness in relation to the concept of SecDevOps. We deploy our security management architecture with Bitnobi [1], a privacy-guaranteeing big-data management platform. We are interested in two specific scenarios that arise during software development, known issues with no current mitigation mechanism and unknown issues to the developer. In this case study, the architecture can be used as a solution for the developers and security analysts for these two issues. By using Business Rules to specify security conditions and actions, we enabled two different teams to have a common understanding of the problems and solutions. The developers were able to write their own rule-based patches when they were aware of a vulnerability in the system. With Bitnobi, there was no current validation mechanism to protect against Brute Force or DDoS attacks. With our

[1] http://www.bitnobi.com/

solution, we allowed the developers to protect against these such attacks while they can work on a patch or validation mechanism. In addition, the developers artificially introduced XSS vulnerabilities in Bitnobi that our solution was able to detect and remove. Our case study shows that we are able to detect vulnerabilities more effectively, execute adaptive actions to mitigate the vulnerabilities and notify the teams, thus creating a stronger secure web application. We are able to seamlessly keep the system in constant monitoring and maintenance while the developers continued working on their application. This contribution addresses the first, third and fourth research questions through implementing the platform with a production application. We observe the developers of the application creating security business rules with the platform, detecting application-specific vulnerabilities and running adaptive actions against attacks.

- Throughout this research work, we designed evaluation experiments for our architecture. The purpose of these experiments were to evaluate the limitations of security tools and compare these results with our architecture. We evaluated single sensors versus distributed sensors to observe packet loss from incoming traffic. We implemented the architecture with the distributed sensors to detect a DDoS attack while we increase the traffic demand. We also used a Web Scanner to exploit vulnerabilities, and looked at our architecture's ability to detect the attacks. The results from this evaluation shows the architecture's effectiveness with regards to vulnerability detection. Additionally, we evaluate the performance overhead to the end-user when the architecture is protecting the application and propose technical solutions that can decrease the performance overhead of our architecture. These include moving reads and writes to the RAM and using Secure Shell File System (SSHFS) to separate the network streaming tool from the web application. This contribution begins to validate the research questions with the listed experiments.

1.4 Thesis Organization

This research work is structured as follows. Chapter 2 presents the background and research related to the field. Chapter 3 presents the framework of the architecture, properties of our solution and the implementation. Chapter 4 demonstrates the architecture in a variety of different security and application use cases. Chapter 5 applies this architecture to a data-intensive industrial application and present the findings for our case study. Chapter 7 discusses the findings of our experimentation with the architecture. Finally, Chapter 8 concludes our work and presents the next steps to improve the architecture based on our research.

Chapter 2

Background and Related Work

To understand the research work being presented, key concepts of the research field have to be presented. In this chapter, we introduce the background of our research field and underlying concepts behind our platforms design. We then discuss relevant work that other researchers have been working on in this research field.

2.1 Background

2.1.1 Cloud Computing

To understand the importance of distribution and hierarchy, it is first important to understand the importance behind Cloud Computing. The US National Institute of Standards and Technology (NIST) defines Cloud Computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [12]. This characteristics of sharing computing resources from a large network provides us with many advantages. We are able to self-provision resources on demand, access them from anywhere, resources can be pooled where multiple users can be served on a single physical hardware and we are able to scale the resources based on the demand. These characteristics contribute to increased performance and reduced costs [13].

There are three main service models provided by the service providers used in Cloud Computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS provides the user with the computing and storage resources in the form of Virtual Machines. PaaS gives the user the capability to develop and deploy applications by providing development tools, APIs, Software Libraries and services. SaaS provides the user with the complete application or the user interface to interact with the application [13]. In our research work, we use the IaaS model, provisioning multiple virtual machines from the service provider to build and use our architecture. NIST [12] also defines the cloud deployment models: Public Cloud, Private Cloud, Hybrid Cloud and Community Cloud. Public Cloud is the collection of cloud resources and services available to the general public and any organization. Private Cloud is the collection of resources and services available only to a specific organization. Hybrid Cloud is the combination of the two, and Community Cloud is a cloud service provided by multiple organizations with the same policies and compliances. In our research, we use a community cloud with physical hardware hosted by several universities available to researchers.

We also incorporate the MAPE-K Loop in our proposed architecture. The MAPE-K control loop is a model that many autonomous systems use as an autonomous manager [14]. The Monitoring component is responsible for capturing data and metrics and the Analyze component processes the data that is captured. The Planning component is responsible for selecting the actions that need to be applied, and the Execution component applied these actions on the system. The K is the Knowledge base of the MAPE loop and contains the data of the environment, the adaption goals and any relevant states related to the MAPE components [15]. With our architecture, we use the MAPE-K Loop to monitor incoming security data and executing adaption actions against these security threats.

2.1.2 Microservices and Containerization

Traditionally, software systems are developed as a single large program which is to perform all its required tasks. Using this traditional approach, we are only able to scale vertically, i.e. depending on the demand, we scale the hardware resources of our server. However, there may be issues with this approach. Scaling vertically can be expensive because of additional hardware resources. There will also be downtime on the application due to hardware changes when scaling vertically [16]. Instead of using a single large program, we can take advantage of Microservices, an important concept that is key to our architecture. Microservices architecture allows software systems to be developed and composed into small, independent components that interact with each other over the network [17]. Using independent components as Microservices, we are able to scale horizontally, i.e., by adding more servers rather than increase a single server's resource. One of the main drivers of the Microservices concept is containerization. Containers are the encapsulation of a component or application with all of its dependencies (binary files, libraries, configurations, etc.) [18]. One key idea to note is that containerization and Microservices are not directly related. Microservices architectures can be created without the use of containers, and containers can be used as a monolithic application [19]. However, the use of containers can support the concept of

Microservices. We are able to separate the components of an architecture into multiple containers and spread them out through the network.

Running Microservices components in multiple independent containers offers many advantages. Instead of running the application or components directly on the OS, containers share the resources of its host OS, and there is little to no overhead compared to a native application. Containers are portable, thus being able to run in any environment. This reduces any bugs that may arise from an environment change and speeds the process of deployment. Since containers are self-composed services, they can also be easily changed, deleted and deployed on-command without any additional set-up. Finally, multiple containers can run on the platform, each as independent components, complex architectures and frameworks can be deployed easily [18] .

2.1.3 Intermediate Language

The intermediate language in this work is the BRE (Business Rule Engine), which makes use of business rules. A business rule is a statement that defines or constrains some aspect of the business. These rules are essentially When-Then statements that dictate the requirements of a business. For example, an insurance company is giving car insurance to their customers. Their business requirements is that customers under the age of 25 pay higher insurance. The business rule in this example would be **When** Customer is less than 25, **Then** insurance cost equals an extra 300 dollars. This is a very basic business rules, but with a BRE and it's properties, we can write a rich amount of rules that is able to perform more complex requirements. The two main characteristics of business rules are that they concern the structure and the behavior of the business [9]. In our case, the business is the application specifications. The BRE comprises the software components that allow analysts, developers and non-programmers to manage the rules, by adding, removing or changing rules when necessary. BREs have been successfully used in many domains, such as finance, healthcare, retail, manufacturing, marketing and other industries [9]. In this research, our focus is the use of BREs as an intermediate language in a security, maintenance and autonomous context.

2.1.4 Streaming Analytics

The architecture we propose uses a Big Data processing analytic system. The term Big Data has various definitions proposed by different researchers through the field. The most simple definition is that Big Data is a large collection of data. For the industry, this would be terabytes and petabytes of data [20]. Other than volume, other definitions define Big Data as the richness and variety of data [21]. Big Data can be classified as structured, unstructured and semi-structured. Structured data has a defined organizational structure and can be usually represented by a schema [20]. Unstructured data does not have any predefined organizational schema [20]. Semi-structured uses the combination of both, the data can be organized but can also have arbitrary associations. [20] Big Data comes from many sources. Huge Data Warehouses may have decades of data to be analyzed. Social media provides a rich, extremely large and unstructured data for analysis. In the security context, application logs, metrics and sensors of large-scale cloud applications can provide a large amount of data to by analyzed for security threats. However, a traditional tool have been built for smaller applications, and may not be sufficient to detect security threats.

A Big Data analytic processing system, specifically Streaming Analytics, is one of the solutions we implement in our architecture. Big Data Analytic are used to find patterns and extract any useful information from large volumes of data. One of the first popular solutions is Hadoop [1], which provides open source software for scalable and distributed computing. MapReduce is one of the central concepts of Hadoop and Big Data Analytics. MapReduce is composed of jobs, the first is the Map Job, which takes the dataset and converts it into another set of data, breaking down the dataset into tuples (key/value pairs). The Reduce will take the output from the Map function and combines the tuples into a smaller set of tuples [22]. The MapReduce model provides a way for parallel processing and allows for a scalable, resilient and flexible way to analyze large datasets. The Analytics solution that we employ is Apache Spark. Spark extends from Hadoop and the MapReduce model, by providing more types of computations. Similar to Hadoop, the Apache Spark Project has many libraries for different needs. The main component is Spark Core, which provides the basic functionalities of Spark such as task scheduling, memory management, fault recovery, interacting with storage system and many other functions. Spark takes advantage of Resilient Distributed Datasets (RDD), a read-only, partitioned collection of records that can only be created through deterministic operations on data in stable storage and other RDDs.

Since we are looking at analyzing a continuous stream of data for security threats, Apache Spark provides a Streaming Analytics component, Spark Streaming. Spark Streaming uses the Spark API for stream processing, making it easy to built a fault-tolerant streaming application [23]. Spark Streaming builds on top of the concept of RDDs and uses an abstraction called Discretized Stream (DStream). Dstream represents the continuous stream of data, created from input data streams or applying operations on other DStreams. Internally, DStreams is represented as a sequence of RDDs [23]. Spark Streaming treats the streaming computations as a continuous series of smaller batch computations. Each of these input batches are then formed into RDDs

[1]http://hadoop.apache.org/

and are processed using Spark Jobs [24]. For our research, we take the functionalities and advantages of Spark and Spark Streaming and make use of it in our architecture. We combine the transformation computation with the Business Rules Engine, and run the batches against the ruleset. This allows us to analyze for security threats while we can scale and distribute the processing power throughout the network.

2.1.5 SecDevOps

DevOps stands for Development plus Operations and is considered a new way of thinking and working, providing a framework for people and teams to be more effective in their craft [25]. DevOps helps different teams align development, testing, deployment and support [26]. DevOps is not a software methodology that follows a strict timeline, but rather a new paradigm that provides tools, values and knowledge to support collaboration between different teams. Some successful businesses that effectively used DevOps in their application include Etsy, Scotiabank and many more. Businesses have begun hiring specialized DevOps engineers to increase effectiveness and share their knowledge of collaboration. Etsy is an online store that allows individual creators to sell their works to people. They implemented a DevOps approach to their development structure, placing a high focus on transparency and monitoring data. Using DevOps and several of its supported tools, they were able to increase the speed of implementation and reduce unnecessary costs [25]. Amazon provides DevOps services and support for AWS, such as Amazon Cloudwatch, AWS developer tools and many more. These help for collaborating code between teams, monitoring data, sharing microservices and creating an infrastructure for a more efficient development pipeline. However, DevOps tends to be implemented without consideration for security, which can result in long-term issues such as vulnerabilities and malicious attacks. To build upon DevOps, another new way of thinking was coined, SecDevOps [7]. SecDevOps incorporates the security team to continuously collaborate with the Developments and Operation teams to build a secure, protection application more efficiently. SecDevOps practices include static and dynamic code analysis, and run-time application protection [8].

2.2 Related Work

Distributing security is an important design decision to consider for many web-based applications and large scale systems. Kruegel et. al [27] support our claim that network-based IDS are barely capable of handling real-time traffic, and a single-node setup will have difficulty in retrieving all packets. They show that architectural and system parameters are often affect the packet droppage of the IDS. One of the earlier works of distributing IDS was by Huang et. al [28], where they proposed a distributed agent-based IDS architecture to detect security attackers. Distributing the IDS and sensors with different programming models and solutions across multiple network nodes has been proposed and designed to improve efficiency, scalability, performance and detection [29, 30]. Vigna et. al [31] describe an approach for distributed intrusion detection, using a network of sensors built around a State Transition Analysis Technique (STAT) framework, which is able to model entire classes of attacks. The proposed STAT-based approach is highly flexible and configurable; the work states that configuration of sensors can be changed in real-time to deal with new and unknown attacks. Our architecture's main focus is not on how to distribute IDS monitors but instead builds on top of IDS distribution. We achieve this by taking advantage of the data distribution and combining it with a decision-making process for the entire system while placing security control in the hands of developers and business analysts. In addition, our architecture adds the ability to dynamically add monitors and new data sources to support expandability during run-time.

Ramsurrun and Soyjaudah [32] designed a cluster security gateway that uses a combination of tools to create a stateful firewall cluster. Messages can be obtained from other firewall nodes and inserted in an internal cache for synchronization among all other nodes. Their design also uses load balancing techniques to distribute the mitigation of a Distributed Denial of Service (DDoS) attack and prevent a system shutdown. With their models, IDS nodes are also used to define new policies for the firewall by informing the policy manager. Ko et al. [33] proposed a distributed recognition and accountability algorithm to observe how a malicious user would move around a network of computers, and designed an architecture that distributes monitoring with a centralized data analysis. However, these solution is primarily focused on the network level with the usage of firewall nodes, whereas our architecture focuses more on the application layer, allowing continuous application-specific development and expansion of the architecture in correspondence with the runtime application protection stage of SecDevOps.

Our primary goal is to take the advantages and practices of distributed security, and use a Business Rule-based platform to help developers understand security at the application-level and collaborate with the security teams. Business rules and BRE have been explored in different contexts to help increase understantability and communication. Jiang et al. used Business Rules to be an intermediate language to integrate electronic forms with a workflow management system. Lojka et al. have used BREs in education where they support virtual laboratories allowing the teachers to easily implement educational methodologies and improves the monitoring, controlling and interactions with their students [34]. BREs have also been integrated with service-oriented architectures using interfaces and service requests for writing rules [35]. This was an early approach for distributed rule execution. Feng and Subramanian [36] present the use of BREs in control center applications and address issues of performance and integration for legacy applications by incorporating remedial action schemes and other design changes. BREs have proven to be effective in these contexts as an intermediate language. In our work, we take the BRE and use it in a security control context to support SecDevOps and security collaboration between teams.

Monitoring is a key component of DevOps and effective log management is a central part of monitoring. An important contributor in log management is Security Information and Event Management (SIEM) [37, 38]. SIEM is a solution that provides central security control in a cloud environment. The basic architecture that SIEMs follow is collecting data from a large selection of source devices, normalizing the obtained data, applying rules and correlating events, storing the log for traceability and auditing. SIEM also provides users with a type of console for administrators or users to access the overall events [39]. IBM provides a SIEMS security analytics tool in a cloud environment, QRadar [40]. QRadar is able to collect log and event data from various sources, such as security devices, operating systems, databases, routers. It normalizes and correlates the data using advanced sense analytics. This allows QRadar to link and identify complex security incidents and threats in the cloud environment. The main differences between our architecture and QRadar is the ability of the former to close the MAPE (Montitor, Analyze, Planning and Execution) loop for security control. QRadar can monitor and analyze log sources and network flow data from thousands of devices, endpoints, and applications, and send the report to the security analyst team to review and take manual action against threats. In contrast, our architecture has the ability to be autonomous, and execute adaptive actions to mitigate attacks and address vulnerabilities. In addition, our architecture can also scale security control while simplifying the ability to define security correlation rules for the developer.

2.3 Summary

In this chapter, we introduced the basic concepts and background for our proposed platform. We then discussed relevant literature related to distributed architectures, intermediate languages in other contexts and security monitoring, and presented the differences with our platform.

Chapter 3

Architecture

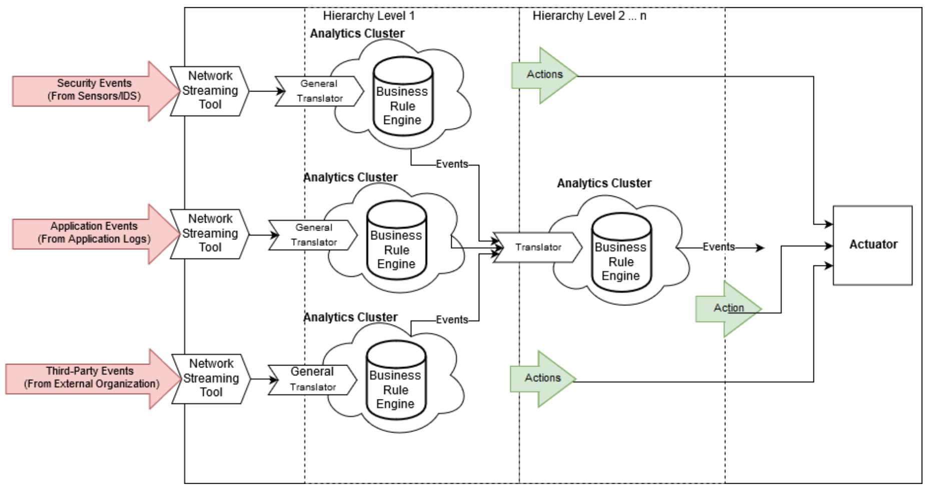


Figure 1: Proposed Architecture for Hierarchical Distributed Security Control

In this section, we describe the underlying architecture of our platform and the data flow between the environment and the application. As seen in Figure 1, the data sources and sensors are placed as the input for the architecture. Data sources can include various logs of the system such as access logs, error logs and application-specific logs. Sensors that react with the environment and create alerts are also important data for the analysts. Third-party data can also be inserted, such as updates from the Common

Vulnerability and Exposure (CVE) database. These data sources are the input to the Analytics Cluster in the architecture. We use a networking utility tool on the application to stream the data into our architecture. This tool allows us to specify the IP address and the port of the cluster to stream the input data.

When the data is streamed into the Analytics Cluster, it passes through a general translator. This general translator parses the raw data into segments making it easier to digest for rule writing and analysis. The Analytics Cluster makes our platform scalable and hierarchical, allowing for large rulesets, distributed responsibility, handling multiple data sources and easier rule maintenance. In Figure 1, each cluster is organized to be responsible for a specific type and source of data, thus splitting the responsibilities. The role of the clusters is to primarily aggregate the partial states provided by the data source to make decisions and take actions on the overall system from different levels in the hierarchy. Each level creates events from it's specific ruleset, and pushes the events to the next level of the hierarchy, allowing the architecture to make very complex decisions. To address scalability, our platform can easily expand by adding more worker nodes to the clusters to deal with larger amounts of data.

With respect to intra-team communication, the security analysts may not completely understand the application-specific logs and the developers may not comprehend and employ security best practices for their application. To address the communication issues, we introduce the Business Rules Engine (BRE) for each of the Analytics Clusters as the platform that aggregates all data from multiple sources and the teams can work together to write business rules that get triggered when a specific requirement is met against all the data sources. Business rules are statements that define or constrain some aspect of the business and consists of "When - Then" statements. Business rules operate as follows: *When* a condition occurs, *Then* perform a consequence or action.

We can imagine the DevOps rule creation process as a library or web service. The security team provides the specification as security policies and the BRE provides a Software Development Kit (SDK) in Java with which the development team can write rules to enforce policies.

Traditionally, each of the sensors is responsible for a subset of the network traffic to prevent saturation and there are multiple logs with different types of data, creating the issue of impartial information and a major gap in security. The architecture is then able to distribute the BRE ruleset for a specific security problem or a specific data source between different clusters, shown in Figure 1 with each cluster holding a Business Rules Engine. This allows us to eventually be able to manage an increasing number and complexity of rules with the same efficiency. The clusters are designed by the organization and depends on the requirements of the application, and the data sources available. The output from the analysis from clusters at the lowest level of the hierarchy can be streamed into clusters at the top of the hierarchy. This enables the architecture to aggregate the events of the system and to correlate data from multiple sources. Figure 1 shows two clusters streaming to the cluster at the top of the hierarchy allowing complex analysis for multiple types of data. For example, multiple security events that were created from the lower levels in the hierarchy are streamed into the higher-level, which aggregates these events to find a system-wide attack. In addition, with a distributed ruleset, it becomes easier to trace rules and events, and understand the business impact.

Our BRE is based on the Production Rule System (PRS) [41] which is shown in Figure 2. The BRE stores the rules in the Production Memory. Incoming and declared objects from the data source are stored in the Working Memory. The Pattern Matcher matches the objects from the Working Memory against the business rules in the Produc-

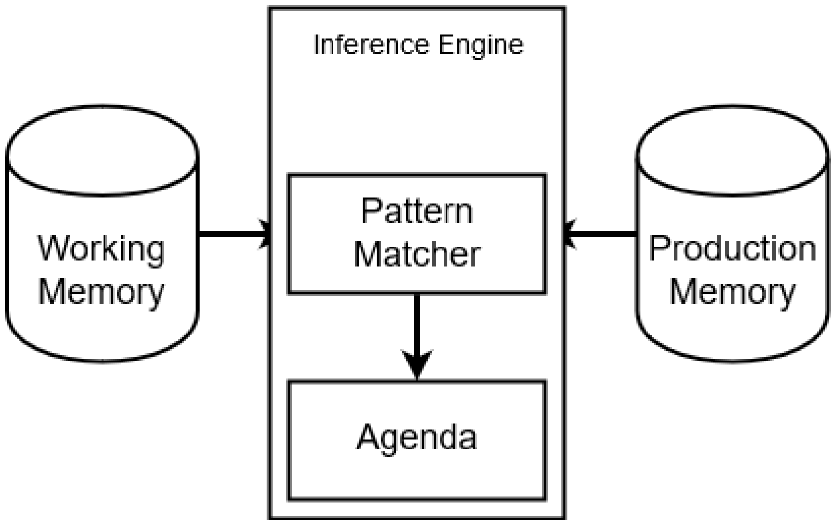


Figure 2: High-level view of a Production Rule System

tion Memory. This is performed by the Inference Engine. The Agenda is responsible for execution order of the rules and any conflicts between the rules [41].

Since the BRE is stateful, the inflowing data can be stored as objects in the BRE Working Memory. This enables the security analysts or the developers to create complex chains of rules with the simplicity of business rules. Chaining is a method of execution for a rule system and can have two forms: Forward Chaining and Backward Chaining. Forward chaining is when the incoming objects trigger a sequence of rules alongside the existing objects in the Working Memory and the BRE finds a particular conclusion at the end of the sequence. An example in a security context is when an attack occurs or a suspicious behavior is detected, the rule inserts it in the Working Memory as an event. This event can be applied as a condition of a new rule to create another event. This can be repeated until the BRE detects an attack. Backward chaining can also be used: once a rule is triggered, all the objects in the Working Memory responsible for the rule can be retrieved for specialized analysis by the teams. Another benefit of the stateful BRE is that previous decisions and suspicious data can be kept as a state in the Working Memory for future use, when complex security attacks arise. Historical information can be correlated with incoming streaming events, which can be indicators of compromised data or data breaches. The BRE provides temporal operations which can be used in the condition to create complex time-ordered rules. For example, if event A occurs 10 seconds after event B, alert the system. This can be taken advantage against security threats that occur in different order or a specific time frame.

With BRE, the development or security teams can specify an adaptive action once the rule is triggered. Each BRE cluster has the ability to execute an action, as seen in Figure 1. Since an application can be hosted on a large-scale network using multiple machines, our architecture includes an actuator to perform actions against the system based on the BRE decisions. As of currently, the actuator is a single generic component that is responsible for all the adaptive actions and uses built-in libraries to execute these actions. The BRE sends the specified action and values required for the action to execute (e.g. The BRE sends the IP Block action along with the IP address to block) to the actuator. These actions include IP Blocking, email notification, changes to the application and changes to the database. The actions completely depend on the system and the nature of the attack.

3.1 Properties of the Architecture

In this section, we list the properties of our architecture and how they are an advantage to security control.

3.1.1 Scalability and Hierarchical

Our architecture is required to be scalable and hierarchical for distributing a large ruleset, handling multiple data sources and to make management of the rules easier. With a single Business Rules Engine, large ruleset processing the continuous stream of events may cause performance issues. With the huge amount of known security problems, vulnerabilities and potential software errors, the ruleset is expected to grow continuously throughout the development process. In addition, the larger the ruleset becomes, the more difficult it is to navigate through all the rules and event chains. Another issue arises when we begin to add and distribute our data sources and sensors. As Silberberg [42] explains, when a system becomes more distributed, the behaviour become increasingly difficult to visualize and describe. When more data sources are added to the system; we lose the ability to understand the entire system state as a whole.

The architecture proposed in this research work is able to distribute the ruleset for a specific security problem or a specific data source between different clusters. For each distributed clusters, our architecture can add more nodes based on the number of rules or events that needs to be processed. In addition, with a distributed ruleset, it becomes easier to trace rules and events, and understand the business rules. Our architecture is able to aggregate data from multiple monitors and data types, making high-level decisions based on the aggregated data.

Distributing the Streaming Analytics Cluster supports the ability to analyze the trace of alerts from the sensors and events from the lower-level clusters in the architecture. The architecture also has the flexibility to add new rules to the business rules engine to address more attacks during runtime without any change to the application. By making use of a Business Rules Engine, this allows business analysts and developers to create new security rules and manage actions of the system.

3.1.2 Minimum Impact on the System

On existing applications, it can become difficult to integrate new functionalities. Casteleyn [43] explains that evolving an application requires starting a new development process by developers who understand the code of the existing application. Integration for new features is time-consuming and costly, and Jacobs [44] mentions that incorporating security measures is not only hard to design and implement, but may decrease efficiency and can make the system unusable.

Instead of integration, the proposed architecture is complementary to the application that it is protecting, and can easily be placed on existing applications without any hindrance or affect on the application's functionalities. This is possible since the only requirement for our platform is a source of data that can be analyzed. Only the networking utility tool has to be deployed on the application or has to access the data in order to stream the data to our architecture which can be placed on a separate virtual machine with no relation or dependencies for the application itself. There is no change in the application or web service required. However, the only limitation is the performance of using the network utility tool which causes a 12 - 18 % overhead on the application.

3.1.3 Flexibility

Our platform is flexible and can easily be implemented with applications from a variety of different types and fields. We implemented our platform with Wordpress, PHP Applications and NodeJS Applications. However, the architecture can be applied to non HTTP-based applications. The only requirement is that there needs to be data that can be fed into the architecture and security threats and irregular patterns can be found through this data source.

3.1.4 Adaptive Actions

Our architecture is able to perform actions based on an event. When a rule triggers, the architecture either makes an action to an internal or external system. These actions include an IP block, logging information and sending email notifications. These actions are sent to the Actuator, which executes this action.

These changes to the environment also change the data flow, and can be used to trigger additional rules in the engine. The ability to change the environment as soon as a rule triggers is a powerful and necessary tool for a secure system. The actions implemented in our platform currently are:

- IP Block
- Email Notification
- Logging Security Events to a specified file
- Database Changes

New actions have to be programmed into the platform, and we are looking at different use cases to find effective and required actions to implement for the future.

3.1.5 Integration with different sources

Our architecture shows its capabilities in security control by being able to process a wide range of data sources. This includes IDS sensors, Database logs, Wordpress plugins, Queries and is able to handle any new data type. The data sources are placed at the lowest level, and the hierarchical architecture creates events when a rule triggers at the low level in the hierarchy. These events are streamed to higher-level abstractions into the business rules engine. Since the rules engine is stateful, we have a clear picture of the entire state of the system from all the data and lower level sources. The architecture can then make complex decisions based on a combination of lower-level and higher-level abstractions as well as a chain of events triggered by multiple rules.

3.1.6 Expandability

New data sources and sensors can be seamlessly integrated into the architecture, as well as new clusters for the data source and its corresponding ruleset. When required, new sensors can be added and the data is set to stream into the ports opened by the architecture. Logs and other data sources can also be streamed to an open port. If another cluster is required for the new data sources, it can be placed during runtime with a new input and output port for streaming. This input port can be the data source or can be another cluster from a lower level in the hierarchy. The output port can be set for the cluster to send data to a higher level cluster in the hierarchy or on standby for future higher level nodes. If needed, the developers or security analysts can take advantage of the ability to update business rules during runtime for the new data sources and sensors.

3.2 Implementation Overview

For the implementation of our architecture, Spark Streaming[1] is used as our Analytics cluster with Drools[2] as the BRE. All data is streamed to the analytic clusters through Netcat[3], a networking utility tool used to forward the log to the architecture. Docker[4] is instrumental of our architecture as it provides a self-contained and lightweight method

[1]https://spark.apache.org/docs/latest/streaming-programming-guide.html
[2]https://www.Drools.org/
[3]http://netcat.sourceforge.net/
[4]https://www.docker.com/

to deliver the services for the analytics cluster and the BRE using containers. A container is the runnable instance based on an image, which is a set of instructions for creating the container [45]. For the architectures data sources for our use cases, Snort was used as our main IDS sensor for alerts as it is one of the most widely-deployed open source IDS, owned by Cisco [46]. Suricata was used as the multi-threaded IDS, exhibiting the architecture's ability to incorporate hybrid sensors.

One of the web application that is being protected by the architecture is a Wordpress server composed of a load balancer, two web servers and a MySQL database. We stream the MySQL slow log data into our architecture to make MySQL related business rules, as well as the Wordpress Audit Log plugin for additional wordpress related data. In our application-specific vulnerability experiments, we used Damn Vulnerable Web Application (DVWA) [5], a PHP/MySQL web application that is known to have vulnerabilities and weaknesses. We use DVWA to test our architecture's ability to detect different types of vulnerabilities.

In our implementation, all our components are containerized and when we need to scale, a new container is created and is integrated seamlessly with all the other components during runtime. Other than the benefit of scalability and expandability for our architecture, Docker containers are an important addition in terms of performance. Chung et. al [47] evaluating Docker containers in high performance computing applications found that Docker containers are able to reduce overhead and are more suitable with data-intensive applications. With the large amount of traffic to be treated against security rules, Docker is an ideal platform for building our architecture.

[5]http://www.dvwa.co.uk/

3.3 Summary

In this chapter, we presented our proposed architecture for security control. The architecture's components was designed in an hierarchy to distribute the responsibilities of the rules and to perform more sophisticated analysis at the higher level of the hierarchy. We then examined the data flow and each of components of our architecture and described their purpose. We explained the BRE, and presented the advantages of using the BRE for a collaborative security control platform. Furthermore, we discussed the properties of using a hierarchical rule-based architecture, and how they are an advantage to the system. Finally, we reported the key technologies and implementations that we used for each of the components in our architecture.

Chapter 4

Use Cases

In this chapter, we implement our platform with common real-world security threats and issues. We demonstrate the implementation of the platform with different types of applications and their architecture. In addition, we show the several sample rules and the data flow of the use case.

4.1 DDoS Detection

The first use case we examined with our architecture was the detection of a DDoS attack. Since DDoS attacks can cause a large amount of stress on the system, Chung [48] identifies DDoS as a scalability problem, not only as a security one. The author argues that instead of content-centric solutions to prevent DDoS attacks, network architectures should focus more on increasing their scalability to better handle sudden surges in incoming traffic and avoid system shutdowns. If a DDoS attack is detected at the lower-level, the Snort nodes create an alert and can take action against the attack as well. However, since the Snort IDS are distributed, the information for each IDS is

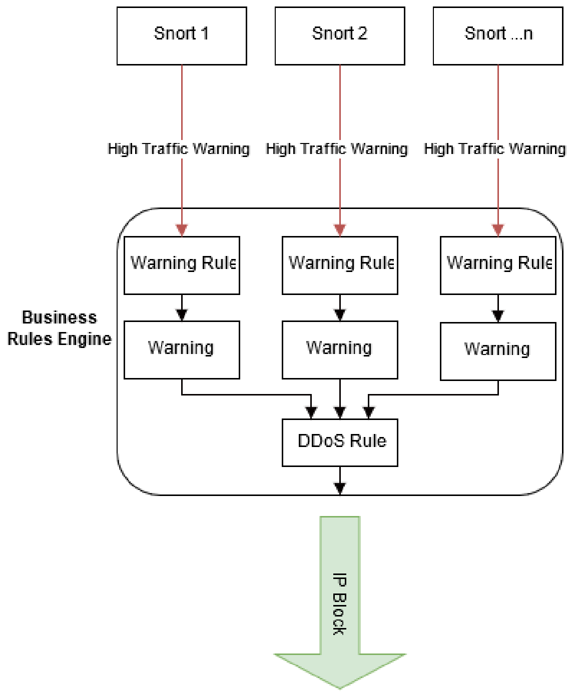


Figure 3: Data Flow for DDoS Detection

partial and not complete, resulting in not enough information for the node to detect a DDoS attack. Our solution is presented in Figure 3, which shows the data flow of the DDoS attack with the BRE. Each of the distributed Snort nodes sends multiple high traffic warnings to the architecture. The Drools engine creates events based on the high traffic warning, and correlates these with the alerts that arrive from all the other nodes. Once the Drools Rules for the DDoS attack is triggered, it uses the IP Block action on the attacker.

In our use case setup, which can be seen in Figure 4, we employed two Snort IDS Nodes as containers for the Wordpress Server as the low level data source. The IPTables is a component that uses IPtables to distribute the request packets of the Wordpress server to each of the Snort IDS. For this Use Case, we developed an Aggregator com-

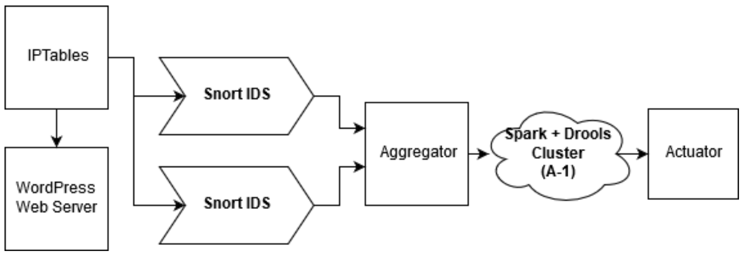


Figure 4: Components Model for DDoS Use Case

ponent, which is a service placed in a Docker container. The Aggregator allows us to identify the sensors in our system and aggregate the alerts from multiple sensors to send to a single cluster A-1. For the Aggregator, we set the IP address and Port of the sensors, and a single output IP address and port for the cluster. Once these are set, the Aggregator will tag the source of the alert which can be used as an identification for the Streaming Analytics Cluster. The Aggregator can add or remove new sensor sources at runtime when needed. The Aggregator then sends the alerts to the Streaming Analytics Cluster. At the start of this use case, we only use a single cluster to run our rules. We deployed an Actuator container which collects the actions from the cluster, and executes it on the system.

At the start of this use case, the user makes requests to the Wordpress Application. The requests first enter through the router entry point, where the users request is forwarded to the web application and duplicated to the IDS nodes for analysis. To set this use case, we used iPerf [1] to emulate regular traffic and Skipfish [2] to inject a DDoS attack in the regular traffic. While Iperf is generating normal web traffic load, we initiate Skipfish on our Wordpress application and begin the attack. The Snort IDS Sensors will analyze the incoming network traffic, and run them against it's own security rules. If the Snort IDS detects an attack, the Snort will be able to act by itself and the alert

[1]https://iperf.fr/
[2]https://github.com/spinkham/skipfish

will be forwarded to the actuator to perform an IP block on the router. However, if Snort is not able to detect the attack, especially in a distributed system where the traffic is divided, the Snort IDS forwards the alert to the cluster A-1, and will be analyzed against the Business Rules Engine. This alert is a High Traffic Warning alert due to the activation of Skipfish during normal traffic.

```
rule "DDoS Attack"
    when
      AlertOne : SnortAlert((
           AlertOne.getMessage().equals("SnortOneHighTrafficWarning") &&
           !AlertOne.getPriority().equals("") &&
      !AlertOne.getIP().equals("")))
      AlertTwo : SnortAlert
           ((AlertTwo.getMessage().equals("SnortTwoHighTrafficWarning")
           && !AlertTwo.getPriority().equals("") &&
      !AlertTwo.getIP().equals(AlertOne.getIP())))
      latestOutput: Output ()
   then
      latestOutput.setOutput("SnortOutput: " + AlertOne.getIP() + ",
           DDoS, " + AlertOne.getPriority());

      Action action = new Action();
      action.ipBlock(AlertOne.getIP(), 20)

      update(latestOutput );
      retract( AlertTwo );
      retract( AlertOne );

end
```

Figure 5: Drools rule for DoS Detection and IP Block

If multiple high traffic warnings appear from all the Snort nodes, then the the DDoS rule is triggered by the Drools engine. A Drools rule sample to detect the DDoS for this use case can be seen in Figure 5. In the When Statement, we first check if the High Traffic warning alert is coming from both our Snort IDS sensors. For example, in case of two Snort nodes, if `AlertOne.getMessage().equals("SnortOneHighTrafficWarning")` and AlertTwo.getMessage().equals(”SnortTwoHighTrafficWarning”) are true, which means this alert occurred on both IDS nodes. In this rule, the IP address of the attacker is also checked to see if they are the same on both machines. Drools then makes the decision that the attack isn't a high traffic warning, but actually a DDoS attempt, and creates a new Action object which is used to block the source IP. Update returns the object back to the Spark Application, and retract removes the alert objects from the working memory.

The next demonstration of this use case was initiating an action or a response to a business rule being triggered. In the Then-statement of the Figure 5 rule, we insert the IP address into the action object, `action.ipBlock(AlertOne.getIP(), 20)`, and block the IP address for 20 seconds. This action object creates the script to sent the IP address of the attackers is sent to the actuator, and the IP address is blocked, effectively preventing the DDoS attack to continue. In the future, we plan on using the architecture to try and prevent Low Slow DDoS attacks, i.e. a DDoS attack that aims to slow a service using extremely slow incoming traffic. Using the business rule engine, we can monitor performance metrics and number of active users. Once the performance metrics do not correspond to a number of active users, the Low Slow DDoS rule should be triggered and will automatically notify the system.

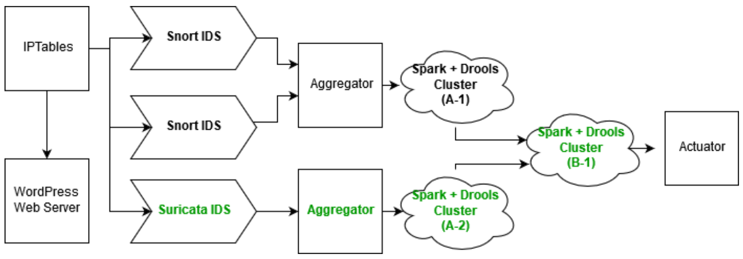


Figure 6: Components Model for DDoS Use Case with Suricata

The architecture is able to add and handle a different type of IDS to compliment the current IDS. Figure 4 shows that we implement another type of IDS for our architecure, Suricata. The previous steps can be repeated, where a Netcat message router and a new aggregator specifically for Suricata has to be added. A new Spark Streaming Analytics Cluster with Drools Business Rule Engine will be added to analyze the stream of alerts from Suricata, and perform necessary decisions. To obtain the complete state of the system and perform complex decisions based on both Snort and Suricata, an Aggregate Spark Analytics Cluster is added. This cluster will read the outputs of both the Snort and Suricata Spark Analytics cluster, and send the decision actions to the actuator to perform. This demonstrates our Platform's ability to be easily flexible and can be implemented with a Hybrid system. This also allows security analysts with different specializations in tools to work together to write sophisticated rules.

```
rule "DDoS Attack"
    when
      snortAlert : SnortAlert
           (snortAlert.getMessage().equals("SnortNodeOneHighTrafficWarning")
           ||
           snortAlert.getMessage().equals("SnortNodeTwoHighTrafficWarning")
           )
      suricataAlert : SuricataAlert
           (suricataAlert.getMessage().equals("(SuricataNodeOneHighTrafficWarning)")
           )
    then
    ...
end
```

Figure 7: Drools rule for Hybrid DoS Detection

Cluster A-1 BRE handles the Snort IDS alerts as demonstrated previously, with the addition of the output being sent to the next cluster in the Hierarchy, B-2, seen in Figure 6. The added Suricata IDS will not send the alerts to the Cluster A-2, which is responsible for Suricata alerts. The rule written in this cluster is similar to the rules written for Snort, but are formatted specifically for Suricata. The events of Cluster A-2 will be sent to Cluster B-2, where the Snort event is in the working memory. This is where the detection for a Hybrid system can come to play. A sample rule for the DDoS detection in Cluster B-1 can be shown in Figure 7, where it checks for a Suricata event and a Snort event in the working memory. If the events from both IDS show that there is a high traffic warning, `snortAlert.getMessage() .equals("SnortNodeTwoHighTrafficWarning")` and `suricataAlert.getMessage() .equals("(SuricataNodeOneHighTrafficWarning)")`, Cluster B-1 will detect this as a DDoS attack and send the action to the Actuator.

4.2 Wordpress Compromised Account

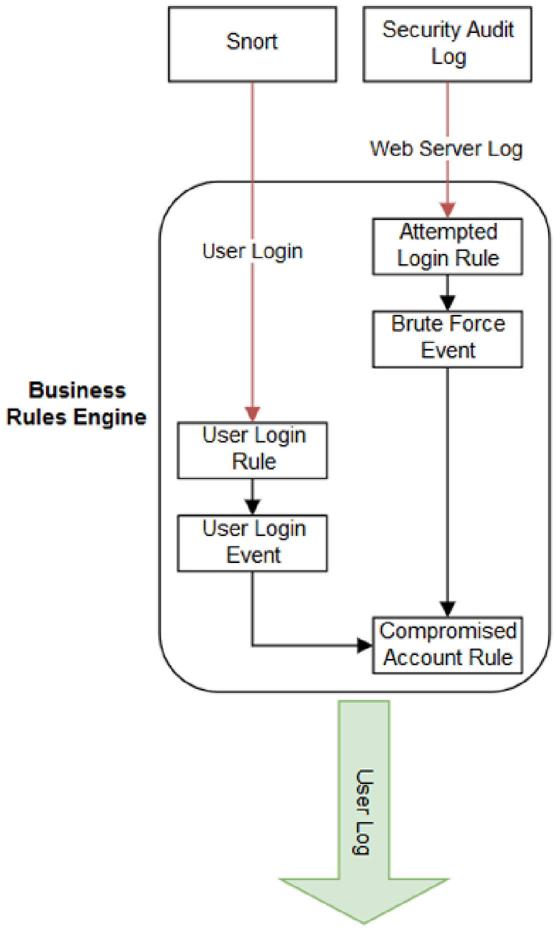


Figure 8: Data Flow for Compromised Account

In this case, we use a combination of the Snort IDS and a Wordpress web application and we try to detect an account that has been compromised. A Wordpress audit log plugin was installed on the applications, which stores a log of users and actions taking place in the Wordpress server into the MySQL database. The dataflow can be seen in Figure 8, where the Snort IDS alerts and the Security Audit Log are streamed into the BRE to detect a compromised account.

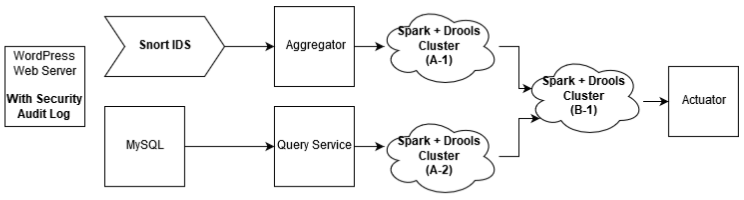


Figure 9: Components Model for Compromised Account Use Case

The set-up can be seen in Figure 9, where we have a Wordpress Web Server with a Security Audit Log installed and a Snort IDS deployed. The Security Audit Log is stored in the Wordpress MySQL server. We created a container that queries the Audit Log in time intervals, and streams newly added rows to the A-2 Web Cluster. The Snort IDS follows the same set-up as the DDoS Use Case. The events created in A-1 and A-2 Cluster are streamed into A-3 Cluster for further analysis.

```
rule "Admin Brute Force"
    when
       a : WSAlert ( ( a.getAttempts() >= 8 && a.getUserName() ==
            "admin" && a.getOutput() == "" ) , myWSAlert : userName)
    then
      a.setOutput( "MySQLOutput: "+ a.getOccurrenceID()+ ", " +
           a.getUserName()+ ", " + a.getClientIP() + ", " +
           a.getAttempts() + ", Brute Force Attempt");
      update( a );
end
```

Figure 10: Drools rule for Security Audit Log

In the A-2 Cluster, the first rule we created was to detect a Brute Force attack on our Wordpress application as seen in Figure 10. We check the username of the login attempt and the number of login attempts. If the username is admin and the number of attempts is over 8, we can begin to suspect that there is a Brute Force attack occurring against the admin user of the application. Since Brute Forces are a common occurrence in many web services, it might not be a major concern for the system. The Brute Force Attacks are only logged in the Drools knowledge base and the events are transferred to the Cluster B-1 in the hierarchy for further analysis.

```
rule "Create Output"
    when
      completeAlert : SnortAlert (
          (!completeAlert.getPriority().equals("") &&
          !completeAlert.getMessage().equals("") &&
          !completeAlert.getIP().equals("") &&
          completeAlert.getCurrent() == true) )
      latestOutput: Output ()
   then
      System.out.println("Creating Complete Output");
      completeAlert.setOutput("SnortOutput: " + completeAlert.getIP() +
          ", " + completeAlert.getMessage() + ", " +
          completeAlert.getPriority());
      latestOutput.setOutput("SnortOutput: " + completeAlert.getIP() +
          ", " + completeAlert.getMessage() + ", " +
          completeAlert.getPriority());

      System.out.println(completeAlert.getOutput());
      // Complete Alert Reset
      completeAlert.setCurrent(false);

      update( latestOutput );
      update( completeAlert );
end
```

Figure 11: Drools rule for Snort Event Creation

At the same time, the Snort IDS is also analyzing the network traffic and creating alerts. This alert if fed into the BRE and ran against the rule seen in Figure 11. This rule is not used for detecting any attacks, instead the rule collects all the parameters of the Snort alert and creates an output. This output is sent to the Cluster B-2, where Figure 10 rule's output was also sent for analysis.

```
rule "Compromised Account"
    when
      sqlAlert : WAlert (
          (sqlAlert.getMessage().equals("BruteForceAttempt") ) )
      snortAlert : SnortAlert
          (snortAlert.getMessage().equals("AdminLoggedIn") )
      actionOutput: Action ( (actionOutput.getOutput().equals("") ))
    then
      actionOutput.setOutput( "Compromised Account: " +
          sqlAlert.getUserName() + ", " + snortAlert.getIP() );
      System.out.println("Compromised Account");
      update(actionOutput);
      retract(sqlAlert);
      retract(snortAlert);
      retract(actionOutput);
end
```

Figure 12: Drools rule for Snort Event Creation

Now that both clusters have sent their output to the cluster at the top of the hierarchy, we can now detect more sophisticated attacks. The rule shown in Figure 12 was written for the B-1 Cluster. First, this rule waits until there is both an alert from the Audit Log and from the Snort IDS. These events are streamed from the lower clusters as show from the previous two rules in Figure 5 and Figure 11. At times, many incoming events are unrelated to the rules condition and would either be stored in the working memory or retracted. However, once the Brute Force event and the Admin Login event are streamed into the cluster, they trigger the conditions of the rule, `sqlAlert.getMessage() .equals("BruteForceAttempt")` and snortAlert.getMessage() .equals(”AdminLoggedIn”). This tells the system that there was a Brute Force attack on the Admin user followed by an admin login. This combination can be seen as a compromised account. The compromised admin account rule is triggered, the compromised admin and the corresponding information about the attack is logged, and the whole system is notified. The security team will then take the subsequent steps to respond to this breach.

4.3 MySQL Data Breach

As discussed before, data breaches are a huge problem in business causing compromised data and reputation loss of the company. In this case, we used MySQL slow log to send all queries, the user and IP address requesting the query, rows examined, rows sent and the performance time of the query to the architecture. The idea of this use case is that many organizations have policies that should not be violated.

```
rule "SQL Injection"
    when
      sql : SlowQueryLog(sql.getRowsSent() >= 50)
      latestOutput: Output ()
   then
      ...
end
```

Figure 13: Basic Threshold rule

There will be a threshold rule in which developers or security analysts will define a maximum number of records for specific SQL queries, and define query patterns for specific web request. A sample rule can be seen in Figure 13, where we continuously check the number of rows sent to the client of a MySQL server. The business rule engine will examine the affected number of records from the MySQL server. If the threshold value is violated, the architecture will send the user and IP address to the actuator, which will block the IP address using iptables. Since business rules can be changed at runtime, we can constantly change threshold values depending on the security policies of an organization.

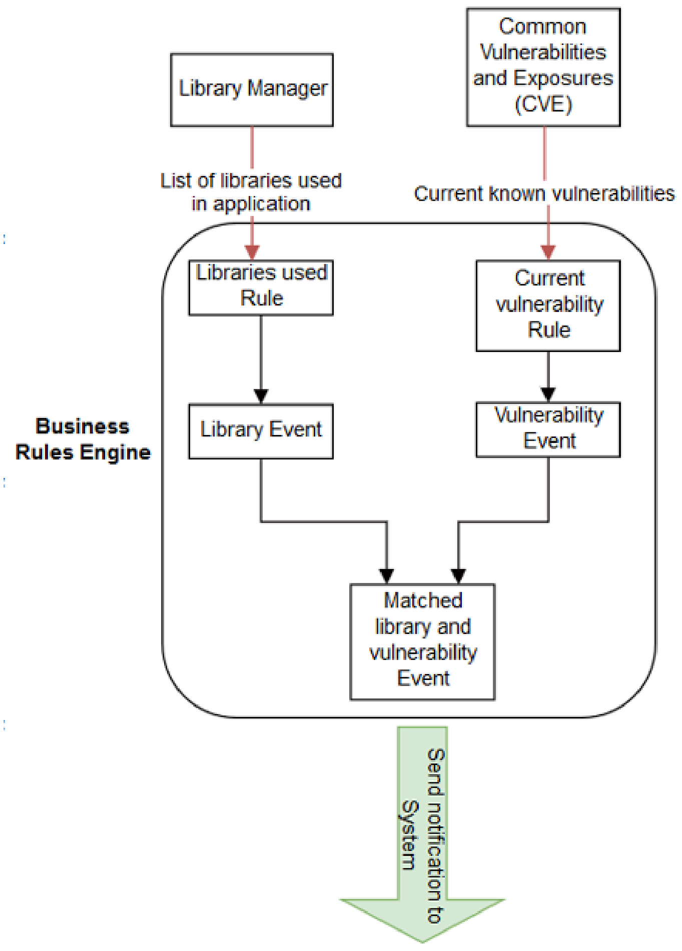

Library Manager
Common Vulnerabilities and Exposures (CVE)
List of libraries used in application
Current known vulnerabilities
Business Rules Engine
Libraries used Rule
Current vulnerability Rule
Library Event
Vulnerability Event
Matched library and vulnerability Event
Send notification to System

Figure 14: Data Flow for Library Vulnerabilities

4.4 Library Vulnerabilities

Software development has become fast-paced and software is continuously evolving. As a result, many developers are required to be efficient when they create web applications. This is solved thanks to the vast amount of open-source libraries available to developers on the web. The developers can make use of these libraries in their applications to easily add frameworks, functionalities and other services that fit their needs without creating them from scratch. However, vulnerabilities from the libraries may be unknown to the developer and can open the software to serious risk. Many of these vulnerabilities are found and recorded by The National Vulnerability Database (NVD). The NVD keeps a list of the Common Vulnerabilities and Exposures (CVE) in technologies and provide a data feed for developers or security analysts. Using our platform and it's stateful properties, we can write rules to detect whenever there is a known vulnerability from the CVE in the developers application.

Figure 14 shows the data flow of this use case, first a library manager, such as Node Package Manager(NPM), will send a list of the libraries and vulnerabilities used to in the application to the platform. Using the stateful capabilities of the business rules engine, these libraries will be stored in the knowledge base. The architecture will then monitor the data feed of the CVE. When a vulnerability in the library used by the application is disclosed, the architecture will notify the developers or the administrator about this new vulnerability in the Software.

```
rule "Check Vulnerability - Product Name and Version"
   when
      vdata : VulnerableData(vdata.getComplete() == true && getProduct()
          != null)
      ldata : LibraryData( vdata.getProduct().contains(ldata.getName())
          && vdata.getProduct().contains(ldata.getVersion()) &&
          ldata.getVulnerable() == false)
      latestOutput: Output ()
   then
      ldata.setVulnerable(true);
      System.out.println("Vulnerability Detected! Match with CVE: " +
          vdata.getProduct() + ", with Library Data: " + ldata.getName()
          + ", " +ldata.getVersion());
      latestOutput.setOutput("Vulnerability Detected! Match with CVE: "
          + vdata.getProduct() + ", with Library Data: " +
          ldata.getName() + ", " +ldata.getVersion());
      update(ldata);
      update(latestOutput);
end
```

Figure 15: Drools rule for Vulnerability Checking

The sample rule in Figure 15 is to check the Vulnerability after the NPM data and the CVE data are both in the working memory of Drools. `vdata.getProduct()` `.contains(ldata.getName())` in the When-statement checks the name of the the package in the CVE and the NPM to see if there is a match, and then `vdata.getProduct()` `.contains(ldata.getVersion())` checks if the version number is similar. If there is a match for both Product and Version number, a library vulnerability may be in the developers application.

4.5 Software Evolution

Software is constantly evolving, adding new functionalities or improving efficiency or previous functionalities. During each new version, there is a potential of new bugs and issues that can be initially unknown. The architecture is able to easily handle software evolution and validating any data. In this case, we used a random application on Github to demonstrate the architecture handling evolution and new functionalities. In our evaluation, we used Gotty, a terminal sharing software. We set up an older version of Gotty where we first set basic Drools rules for validating the users connecting, disconnecting and successful application exit. Then we installed the newer version, with two new functionalities: setting a maximum number of users and a timeout for exiting. In our example, we set threshold rules validating the maximum number of users and the timeout. We then stream the output of Gotty to our architecture, and if the new functionalities are within the threshold, the application is running properly. This is done without changing anything in the source code or stopping the Drools and Spark analytics system, and instead the developer and security team are able to change the Drools Rules at runtime to account for the new functionalities.

4.6 Application Specific Vulnerabilities

Vulnerabilities that arise from issues in the applications code can often be more difficult to identify. These vulnerabilities may be improper state changes, where an unauthorized user can find an access to the admin's portal or application webpage updates by the user without proper validation. With our architecture and the use of DevOps for communication, it is simpler to create specific business rules to detect vulnerabilities that are application specific.

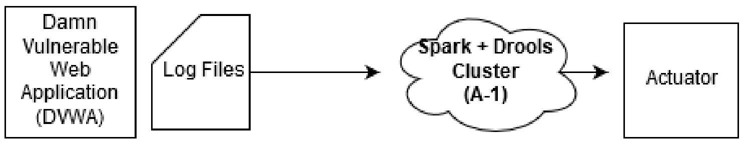

Damn Vulnerable Web Application (DVWA)
Log Files
Spark + Drools Cluster (A-1)
Actuator

Figure 16: Application Specific Components

The set-up components of this use case can be seen in Figure 16, where we used DVWA and streamed the log files with Netcat to the Cluster. We activated the MySQL Slow Log and the Apache2 Access Log and fed it into the Business Rules engine. In addition, we created custom PHP logs for user form entries and interactions for specific web pages. With a combination of these three logs, we created application specific rules to protect against attacks. These rules written were to protect against Cross-Site Scripting (XSS) attacks, SQL Injections, PHP backdoor, Path Traversal and Remote File Inclusions.

A sample XSS rule for the application can be seen in Figure 17. The condition `user.getQuery() .contains("guestbook")` is to check if the attack is targetting the guestbook, a webpage and table that is specific to DVWA. We then check if the query has the keyword `<script>`, which means a user is trying to place a script into the guestbook. However, this can be other XSS-related keywords such as `<body onload` or even the closing of a tag, `/>`, which shouldn't be placed inside the guestbook. For

```
rule "XSS Exploit 1"
  when
    user : SQLQuery (user.getManipulation() == "INSERT" &&
        user.getQuery().contains("guestbook") &&
        user.getQuery().contains("<script>"), this before[1ms] index)
    latestOutput: Output (latestOutput.getOutput() == "")
  then
      ...
end
```

Figure 17: XSS rule for DVWA

this use case, we have multiple rules targetting different types of attacks that require some background of the application.

To attack DVWA, we used ZAP scanner [3], an open-source web application security scanner that can be trained to understand the applications behaviour. This is done by opening the browser with ZAP, and exploring the website and the forms. Once this has been completed, we being the ZAP active scan to find vulnerabilities. An active scan is an attack on the web application, listing the vulnerabilities after the attack. After the ZAP scanner has finished running the dynamic scan, it provides us with the a number of vulnerabilities in DVWA. We then begin a manual penetration with these vulnerabilities, and target and attacked these vulnerabilities individually. When the attack occured due to our manual penetration, the Drools Engine was able to detect an application specific misbehaviour or current attack, such as an XSS attack or a MySQL Injection. In the experiment section, the results can be seen from our vulnerability table.

[3]https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

4.7 Summary

In this chapter, we presented our platform in various real-world security use cases. The uses cases we implemented our platform with were DDoS Detection, Compromised Accounts, MySQL Data Breach, Software Evolution and Application Specific Vulnerabilities.

We introduced the different data flow, the set-up of the components, application and platform of each of the use cases. Furthermore, we presented sample rules for the use cases, demonstrating the rule logic, the writing process and the simplicity of When-Then Statement. We demonstrated how our architecture can aggregate different data sources, handle hybrid sensors, make sophisticated decisions, run adaptive actions, handle third-party data sources, detect application-specific attacks and various other advantages of using our platform.

Chapter 5

Industry Case Study

While understanding the system in a controlled manner is important, we are more interested in examining its application on an actual development and production environment. In this type of environment with different developers working on an application, we can observe the concept of DevOps and SecDevOps in action. The purpose of this case study is to see if and how we can successfully apply our architecture on a data-intensive industrial application. As our architecture is a platform for SecDevOps, we can observe if the the process of dynamic addition of rules are simple to understand by both analysts and developers. Our main argument is that this collaboration should result in a stronger and more secure system.

In the context of our experiments, we deployed our security management application with Bitnobi [1], a privacy-guaranteeing big-data management platform. Bitnobi fits perfectly our case study for a data-intensive application with high requirements in security and privacy, given that it handles large amounts of financial data, like banking and credit records. Bitnobi's deployment is also distributed, which means that all

[1]http://bitnobi.com

problems around volume and distribution of the monitoring data are present. The main idea behind our case study is to have Bitnobi teams (development and security) to use our system to deal with a variety of known and unknown vulnerabilities and security issues. More specifically, there are two scenarios we are interested in examining in the context of continuous security maintenance and evolution. These scenarios are:

1. The BRE detects a vulnerability in the application that the developers do not know about and the developers in response, begin correcting that vulnerability.
2. The developer knows that there is a vulnerability in the application, writes a rule with a corresponding action and uses the BRE to temporarily patch this specific vulnerability.

In the first scenario, we assume that everything (BRE and the application) is already deployed together and there are already prewritten rules based on known vulnerabilities, suspicious user activity and incorrect state changes (e.g. regular user with access to the admin page). When the application is continuously evolving, a new vulnerability may arise that may match these rules. This first experiment shows us that BRE can catch these vulnerabilities as soon as they appear in logging data. This is an example of the dynamic application analysis and runtime application protection of SecDevOps where the architecture monitors and receives feedback once the application is deployed. Vulnerabilities, which may have gone unnoticed during development time when simply looking at the code, are now revealed through malicious traffic at runtime. Besides identification, the BRE can handle this malicious activity, for example, by blocking specific IPs or dropping particular packages, until the vulnerability is fixed at its root cause.

In the second scenario, the developer may have found or already know of a vulnerability but there is no rule in the BRE to protect against it. The motivation behind not fixing the vulnerability immediately could include time or cost constraints. Instead, the developer can write a simple rule to address this vulnerability and dynamically insert in the BRE without shutting down the system. In this experiment, we show that this rule can be very simple and tightly connected to the application itself for the developer to immediately add it as a response to the threat. Once the rule is placed, the developer can notify the security analyst to verify, validate and expand on these rules, further enhancing the security patch of the system. This communication between teams using the BRE represents the DevOps practice of the project.

5.1 Industry Implementation

For the subject system to be managed by the proposed architecture, we collaborated with the Bitnobi team. Bitnobi is a web application already in production that eliminates the need to create copies of data and places the responsibility for launching data jobs on the end user by ensuring that the end user meets the data provider's ”rules of engagement”. Bitnobi is a privacy-protected data sharing platform, which alleviates the issues that arise from the amount of data records that are growing at an exponential rate for many businesses. Business analysts have to sift through these large datasets in order to capture insights for their products and/or services. Data providers also need to copy and mirror the data sets to be analyzed, which may not be secure or anonymized. Moreover, this process may be inefficient when only a segment of the data is required to be analyzed rather than the whole. Bitnobi provides the solution to these issues by allowing data providers to share access to data in a secure manner without releasing raw data or making copies of them. The platform enables the data providers to control access to virtualized segments of the data, allowing the end users to choose the data they need to work with when building a data job, instead of aquiring the entire dataset.

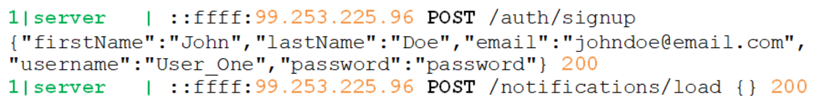
Bitnobi uses large sensitive datasets belonging to business or data providers. This means that Bitnobi itself must be secure from any vulnerabilities and malicious attacks. Bitnobi is also a newly developed product that is constantly being updated every week with new functionalities. For these reasons, we believe that evaluating the SecDevOps platform alongside Bitnobi would be an ideal use case. For the needs of our experiments, we worked with the Bitnobi team on a sandbox copy of the application. Our architecture was placed on a large-sized VM with 8GB RAM, 4 VCPU and a 80.0GB Disk. All components of the architecture were deployed as Docker containers.

Our first step was to identify Bitnobi sources for security monitoring data that could be fed into the Streaming Analytics Cluster. Bitnobi uses NodeJS[2], an open-source, cross-platform JavaScript run-time environment that executes JavaScript code on the server side. In addition, Bitnobi implements a process manager for production applications for NodeJS called PM2[3]. PM2 is able to list all the online components of the application (Web Application, Database, etc), any errors or failed components, monitoring memory and cpu, and log management. In our case, log management was the most important part. PM2 is able to stream all available logs in the web application. Using Netcat, the PM2 log manager streamed all logging information that was set by the developers into the Spark analytics cluster. The general translator was programmed for the Spark Analytics Cluster. It parses the raw data by splitting the string data into segments. This allows the teams to select the specific segment that is required to be analyzed, simplifying the rule writing process. The developer or security analyst can now write rules to correlate the incoming log data from Bitnobi and detect any application-

[2]https://nodejs.org/en/
[3]http://pm2.keymetrics.io/

specific issues.

To detect these application-specific issues, we first need the Bitnobi logs to provide us with enough information to make these types of analysis. The production version of Bitnobi has minimum logging, the only logging information it contained was heap used and the error count. This was not enough information to detect any type of vulnerability within Bitnobi, and we had to make some modifications. The developer version of Bitnobi was also available, but the same issue arose in which we did not have enough information to detect any attacks. We worked with the Bitnobi developers and discussed certain requirements needed in the log for proper data analysis and rule created.



```
1|server   | ::ffff:99.253.225.96 POST /auth/signup
{"firstName":"John","lastName":"Doe","email":"johndoe@email.com",
"username":"User_One","password":"password"} 200
1|server   | ::ffff:99.253.225.96 POST /notifications/load {} 200
```

Figure 18: Bitnobi Logging Output Sample

The developer was already using Morgan [4] to help the development and testing process of Bitnobi. Morgan is an HTTP request logger middleware for node.js and customized important values that can help detect malicious traffic from vulnerabilities. Therefore, we used the Morgan logger for our data source, exhibiting the flexibility of our architecture by implementing it with preexisting logging modules. The developers only had to do simple modifications to log the required information to fully detect and mitigate attacks. The resulting logging structure for the Bitnobi application is: http, IP address, method, url, requestpayload and status. This output can be seen in Figure 18. From the figure, the first section gives the logging data's application origins, in this case, `1|server` is the active Bitnobi application. Logging data can also originate from MongoDB and MySQLReplication. The IP address gives us the source address

[4]https://github.com/expressjs/morgan

where the request is arriving from, which will be important for correlating and blocking attacks. The next part is the method which can be a GET, POST, DELETE request. The url expresses which page is being accessed. The payload of the request shows the request data in a JSON format; any improper or suspicious request can be analyzed and correlated from the JSON requests. Finally, the status shows the HTTP codes, letting the architecture know if the request was successful or a failure. This amount of information is sufficient to begin experimenting with the scenarios and will be streamed into our architecture via Netcat.

```
declare ApplicationLog
   @role(event)
   @ClassReactive
   fullLine: String
   IPAddress: String
   http : String
   method: String
   url : String
   requestpayload: String
   status : String
end
```

Figure 19: Declaring Drools Object

Before we begin creating a ruleset to protect the application, some preliminary declarations and rules have to be created for our achitecture. Figure 19 shows the declaration of a new type; these are akin to objects that will be modified and interacted with in the Drools engine by the rules. In this case, we are declaring the application log that was discussed in the previous paragraph. The `@role` and `@ClassReactive` are property metadata for the type. The `@role(event)` sets the type to be handled as an event which adds the ability of timestamping. The use of timestamps allows us to use the temporal operations such as `Before()` and `After()` which can be used to correlate and combine different events to trigger more complex rules. The `@ClassReactive` is a property that disables property reactivity, which disables (re)evaluation. In our case studies and security-based rules, multiple reevaluations of the events with new events are important and thus, property reactivity has to be disabled. The next lines of the type declarations are the variables that make up the application log, which are the `IPAddress`, `http`, `method`, `url`, `requestpayload` and `status`. To make use of this type, incoming data has to be inserted into the engine as the `applicationLog` object.

```
rule "Insert Application Object"
   when
      rawData : DataInput(
         rawData.getStringData(0).equals("1|server"))
   then
      try{
         ApplicationLog request = new ApplicationLog();
         request.setFullLine(rawData.getFullLine());
         request.setHttp(rawData.getStringData(0));
         request.setIPAddress( rawData.getStringData(2));
            ... // set up of other properties
         retract(rawData);
         insert(request);
      }catch(IndexOutOfBoundsException e){
         System.out.println("Index out of Bounds");
      }
end
```

Figure 20: Drools rule for inserting an application object into working memory

The rule to insert these objects can be seen in Figure 20. The Drools engine then automatically creates getter and setter methods of our declared type which allows us to easily modify the values. The incoming data from Bitnobi is first streamed through a general translator in the analytics cluster, and the data is parsed by spaces and the values are inserted into a list. In the Drools community, Left Hand Side (LHS) is a common name for the conditional part of the rule and Right Hand Side is a common name for the consequence part of the rule. In the rule shown in Figure 20, the Left Hand Side shows that when the first parsed value in the `rawData` equals `1|Server`, which corresponds to the Bitnobi Application log, the log data is then streamed into the declared

`ApplicationLog` type on the Right Hand Side of the rule. Once the values are set, the raw data is retracted from the working memory, and the new `applicationLog` type is inserted for analysis. Since this type is set as an `Event`, it is automatically timestamped once inserted into the Drools engine.

5.2 Scenario One: Unknown vulnerability, existing rule

To setup this scenario, the security analyst alongside the developers had already created ruleset. The developers first had to be trained on how to use Drools. Since Drools is Java-based, it was familiar to the developers and easy to learn as it is closer to application logic. The developers were first presented the architecture, properties of Drools and the basic syntax of the "When - Then" statements. The developers were then shown how to first declare objects, the basic Get and Set functions generated by the Drools Objects and the several built-in functions that Drools provides. Once the developers were able to do some basic exercises using Drools, they were then shown how to interact with the incoming data from Bitnobi. Unlike policies, network and security documentation which developers find difficult to understands, the training process for Drools was simple and straight-forward since it was code-based.

The security analysts provided security requirements, and the developers wrote a ruleset. This included rules to correlate the incoming data and detect any vulnerabilities of Bitnobi. We began this process by navigating through Bitnobi and understanding the normal user activity that arises. Basic rules were created initially. For example, a basic rule checks if the MongoDB and the Bitnobi servers have any errors or connection problems, and if triggered, the rule will notify the developer. Threshold and security-based rules were then added into the ruleset. These included rules for login attempts, Brute Force attacks, checks for broken access controls and cross-site scripting. In this scenario, the unknown vulnerability was cross-site scripting attack (XSS). The Open Web Application Security Project (OWASP) places the XSS attack in the Top 10 application security risks in 2017 [49]. The XSS vulnerability can occur whenever an application includes untrusted data in a new web page without proper validation, or when user-supplied data can be updated into an existing web page using a browser API that can create HTML or JavaScript. XSS allows the attackers to execute their scripts in the victim's browser which can result in hijacking the users session or redirecting the user to a malicious site. Using a combination of rules and actions, the Drools engine can catch and protect against this attack in this case.

```
rule "XSS Attack 1"
   when
      request : ApplicationLog(
         request.getUrl().equals("/users") &&
            request.getRequestpayload()
               .contains(";</script>"))
   then
       Action action = new Action();
      action.notify("Developer", " XSS Error/Vulnerability: See " +
          request.getFullLine());

        action.changeDocument("users", "<script>",
            "temporary-replacement-text");
end
```

Figure 21: Basic Rule for XSS detection written by Security and Development teams

The analyst created some basic XSS detection rules. Figure 21 shows some of the preexisting rules for an XSS attack. On the left-hand side, when the incoming request payload contains a `<script>` or XSS-related tag on the /users page, then the rule is triggered. The analysts wrote the basic keywords to detect an XSS attack, while the developers with their knowledge of Bitnobi, add application-specific triggers in the rules. An example from the figure are `request.getUrl().equals("/users")`, as the developers know the specific pages that may be vulnerable or are new additions to Bitnobi, and haven't been fully tested. Once these rules were set, we deployed the application.

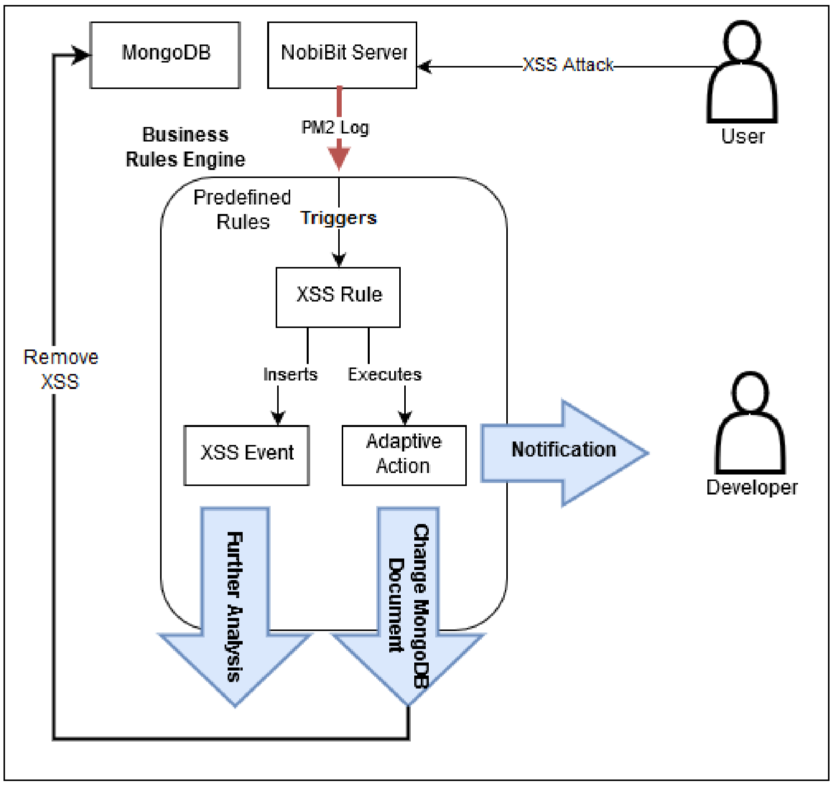


Figure 22: Process of Scenario One for XSS mitigation

During the deployment, attacks occurred during the normal routine traffic where the requests are legitimate. A script was created to automate normal traffic on the web application. The script logs in to Bitnobi, saves the session and uses Apache Bench, a tool for benchmarking, to send a large number of requests to the application. Alongside normal behavior, we manually injected malicious scripts in Bitnobi, seen at the start of the process in Figure 22. In Bitnobi, one of the unsecured XSS vulnerabilities can be exploited when the user edits the profile section: they insert a script into the first or last name form and save it as the user's profile. The next time the user accesses the page, they will be greeted with `"Hello, <script> window.open("http://maliciouswebsite.com") </script> Doe"`. However, this is detected as soon as the user tries to attempt this specific XSS attack by the rules. When the user submits a script, the form request data are logged and streamed directly to the security architecture. Once the data is inserted into the Drools engine, each of the rules is evaluated against this line until the XSS rule is triggered. Figure 22 shows the the process after the rule has been triggered. The rule can store this XSS attack as an event for further analysis as well as perform an adaptive action. One of these adaptive actions is to create a notification. Once the Bitnobi developer is notified about this vulnerability, they can begin working on fixing this issue by adding a sufficient security validation to the edit profile section. Alternatively, once the rule is triggered and the XSS attack is found, an automatic adaptive action, already specified, can be executed. For the XSS rule, the `action.changeDocument()` is used to specify the collection in MongoDB, find the values that have scripting keywords and replace them with a temporary placeholder. Under this continuous monitoring and maintenance system, unidentified vulnerabilities can easily be found during runtime and fixed in a more efficient and time-saving manner.

5.3 Scenario Two: Known vulnerability, no existing rule

Bitnobi is a new application that is used by a specialized set of end users, and the focus for development has been on the client's functional requirements. For this reason, the developers of Bitnobi know that there is limited protection against DDoS and Brute Force attacks in the current version. Our first steps were to see the effects of Bitnobi without any Brute Force protection. For the experiment, the developers purposely misconfigured the middleware allowing attackers to exploit the REST Api to grab the list of all the registered users. Once the attacker obtains the user list, it reduces the complexity of the Brute Force attack. For the Brute Force attacks, we set up THC Hydra[5], a tool that can perform rapid dictionary attacks against a web application. Before the Brute Force attack, we configure the clients containers to send normal web traffic to the application. During normal behavior, we initiated a lengthy Brute Force attack against Bitnobi and there was no responding action against this attack. THC Hydra was able to constantly try different combinations of passwords and Bitnobi kept on logging the increasing number of requests from the Brute Force attack mixed with all the normal login attempts and application access by legitimate users. Without any proper mechanism to defend against a large wave of requests, the Brute Force attack will eventually be successful if the attack goes unnoticed.

This is where our architecture plays a key role in a SecDevOps context. This process can be seen in Figure 23. While the Bitnobi server is still running, and the developer knows that there may be a Brute Force attack in the future because they are aware that there is no application validation system, they begin writing the Brute Force rule. The developer can then create a business rule based on the threshold of login attempts. Once the number login attempts from the same IP address passes the threshold, the rule triggers an adaptive action to temporarily block the IP. Once the temporary IP block is over, the attacker begins the Brute Force attack once again. Since the events of the rules are stored in the working memory, the rule detects the new Brute Force and detects that is has been an IP address that has already been identified to launch Brute Force attacks. From here, the rule's action can be configured to block the IP address for a longer period of time or even permanently. In this scenario, the developer can

[5]`https://github.com/vanhauser-thc/thc-hydra`

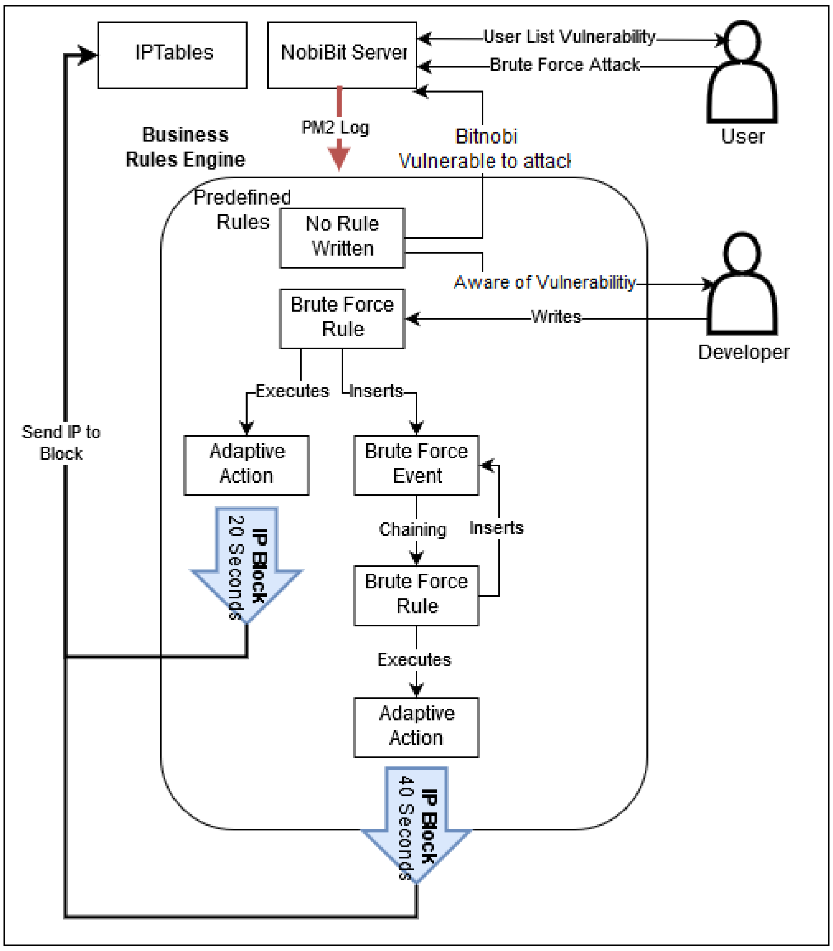


Figure 23: Process of Scenario Two for Brute Force mitigation

support the application's security by adding this temporary patch to fix the breach.

Figure 24 shows the rule that the Developer has written. Drools Business Rule provides a cumulative function, which allows the rule to iterate a function over a collection of objects, returning a result object. For the Brute Force rule, the developer used a combination of `accumulate()` and `count()` on the Left-Hand side, keeping track of all the requests with the same IP address. The calls `attempts.getMethod()`, `attempts.getURL()` and `attempts.getStatus()` are set to check if the incoming requests send a POST request to `/auth/signin`, which results in a bad request, a 400 http code. This part of the rule detects that this request was a failed login. Finally,

```
rule "Brute Force Detection" salience 10
   when
      request : ApplicationLog(
         request.getMethod() == "POST" &&
         request.getUrl() == "/auth/signin" &&
         request.getStatus().equals("400"))

      accumulate ( attempts : ApplicationLog (
         attempts.getIPAddress()
            .equals(request.getIPAddress()) &&
         attempts.getMethod() == "POST" &&
            attempts.getUrl() == "/auth/signin" &&
            attempts.getStatus().equals("400") );
            numLogins : count(attempts) ; numLogins >= 10 )
   then
      Action action = new Action();
      action.ipBlock(request.getIPAddress(), 20);

      Event bruteForceEvent = new Event();
      bruteForceEvent.setName("BruteForceAttack");
      bruteForceEvent.setIPAddress(
         request.getIPAddress());
      insert(bruteForceEvent);
end
```

Figure 24: Drools rule for Brute Force Detection written by Developer

the rule accumulates these objects as attempts, and runs the function count(attempts) and if the threshold is over 10, the rule triggers. The Right-Hand Side shows the `action.ipBlock(request.getIPAddress(), 20)` which sends the IP address to a script that blocks the IP address on IPtables and after 20 seconds, to remove the IP block. The Brute Force event is then created with the corresponding IP address, and inserted into the working memory.

The use of this event can be seen in Figure 25. Once the IP address is no longer blocked, the attacker begins to run another Brute Force attack. However, their IP address is already stored as an event. Once again, Drools accumulates the failed login attempts but now compares it with the IP address of the previously detected Brute Force attack. If that is the case, the rule is triggered and another ipBlock action is executed, blocking the IP for a longer time than previously. Any more additional attacks,

```
rule "Brute Force Detection - Extension" salience 20
   when
      prevBruteForce : Event(
         prevBruteForce.getName()
               .equals("BruteForceAttack"))

      request : ApplicationLog(
         request.getMethod() == "POST" &&
            request.getUrl() == "/auth/signin" &&
            request.getStatus().equals("400"),
            this after prevBruteForce)

      accumulate ( attempts : ApplicationLog (
         (attempts.getIPAddress()
               .equals(request.getIPAddress()) &&
         attempts.getIPAddress()
            .equals(prevBruteForce.getIPAddress())) &&
         attempts.getMethod() == "POST" &&
            attempts.getUrl() == "/auth/signin" &&
            attempts.getStatus().equals("400"),
            this after prevBruteForce );
            numLogins : count(attempts) ; numLogins >= 10 )
   then
      ...
```

Figure 25: Extension Rule of the Brute Force Detection

and this could result in a permanent block. This is an example of the developer being able to chain the rules and mitigate the Brute Force attack by adding a new rule after the application is deployed or even during the Brute Force attack. One keyword to note from both Figure 24 and 25 is Salience. Salience is an important keyword that can be set to define the priority of the execution order of the rules. If multiple rules can be executed without any priorities, they will be executed in an arbitrary order. The higher the Salience value, the higher its priority. In our scenario, the extension has higher priority than the initial Brute Force rule. This is because if the initial Brute Force rule had already been triggered, the developers and the security analysts wanted the extension rule to be checked first to see if the attacker is still in action. If the Salience is not set, then the initial Brute Force rule will be triggered for the same attacker and a longer block would not be given. When THC Hydra is used to execute a Brute Force attack now, it is detected by the rules, the attacker's IP is blocked and the application is secure. This rule can be seen as a temporary patch until the developer adds a proper validation mechanism for the authentication page.

```
rule "User Logged In"
   when
      request : ApplicationLog(
         request.getMethod() == "POST" &&
            request.getUrl() == "/auth/signin" &&
            request.getStatus().equals("200"))
   then
      Event userLoggedIn = new Event();
      userLoggedIn.setName("Logged In User");

      JSONParser parser = new JSONParser();
      Object obj = parser.parse(request.getRequestpayload());
      JSONObject jsonObject = (JSONObject) obj;
      String name = (String) jsonObject.get("username");

      userLoggedIn.setUsername(name);
      userLoggedIn.setIPAddress(
         request.getIPAddress());
      insert(userLoggedIn);
end
```

Figure 26: Event Creation: User Logging In Rule

A vulnerability in REST API allowed the list of all users to be captured. This was intentionally set by the developers to evaluate the Brute Force scenario more effectively. However, this is still a real threat to many web applications and an example of sensitive data exposure, one of the top 10 common vulnerabilities where the API is not properly configured to protect sensitive data [49]. Without the vulnerable REST API, the Brute Force attacker would not have been able to obtain the usernames which would create a more time-consuming attack due to a larger computational complexity. The attacker exploits this vulnerability to use the user list for the Brute Force attack. Using our engine, we are able to detect when the user list or sensitive data have been compromised.

Figure 26 shows a basic rule for creating events of every user who has logged in, and inserting it into the working memory. On the Left-Hand Side of the rule, once there is a successful login (HTTP status code equaling 200) on the authentication page, the rule is triggered. To make use of this event for future correlations, certain values have to be associated with this event. The developer can import external classes in Drools on the Right-Hand Side, in this case, the JSON parser is imported to parse the request payload. Once converted into a JSON object, the developer gets the username. Logging the username itself is pretty essential, but the Event Drools object does not have a username attribute. Here we showcase Drools flexibility, where the developer simply writes `username : String` in the object declaration and now the developer is able to set the username in the rule. They insert this user event into the working memory. The developer and the security analyst can write a rule to detect any potential sensitive data leaks. Using the REST API call to view the list of users should only be available to the developer or the admin of the page, thus the request URL `"/resourcesmanager/listusers"` with a successful http status code is compared with two other conditions. The first condition is when the User event, that was inserted in the previous rule, has the same IP address as the request URL but the user is not an admin. The second condition is when there are no users logged in to Bitnobi currently, but the list of users were still accessed. If one of these conditions are met, then the developer is notified that there is a vulnerability where sensitive data is exposed. The developer now is aware of this issue, and can take action.

5.4 Summary

In this chapter, we implemented our platform with a production application, Bitnobi. We explain the Bitnobi application and the technical implementation specifications with our platform. We presented two scenarios of the application. The first scenario is when the BRE detects a vulnerability that are now known to the developers. This scenario shows the ability to detect threats, and will help developers by providing them the information of what fixes are required. The second scenario is when the developer knows that there is a vulnerability, but there is no rule written in the BRE. The developers, while aware of the threat, may not be able to fix it due to constraints, therefore they write a rule to temporarily detect and patch any exploits to the vulnerability.

We present the data flow of each scenario and then demonstrate the collaboration between the developers and security team. In each scenario, we go through the rule-writing process to detect and mitigate the attacks. Furthermore, we present sample rules for each of the use cases, explaining the components of the rules and the operations that were used.

Chapter 6

Evaluation Experiments

In this chapter, we look at three main experiments related to our platform. The first experiment is the distribution of IDS sensors, where we observe the packet droppage and packet distribution. In the second experiment. we look at using the platform on a web application and observing what vulnerabilities our platform is able to catch. The final experiment is related to the platform overhead on the web application. Specifically, we are interested in seeing the performance effects of implementing our platform and it's required components with the web application.

6.1 Increasing Sensors for Packet Dropping

In these experiments, we view Snort IDS and its ability to handle the number of packets from incoming traffic. We show that increasing the number of IDS sensors will prevent us from losing any valuable information when attempting to detect an attack.

6.1.1 Experiment One: Single Snort IDS

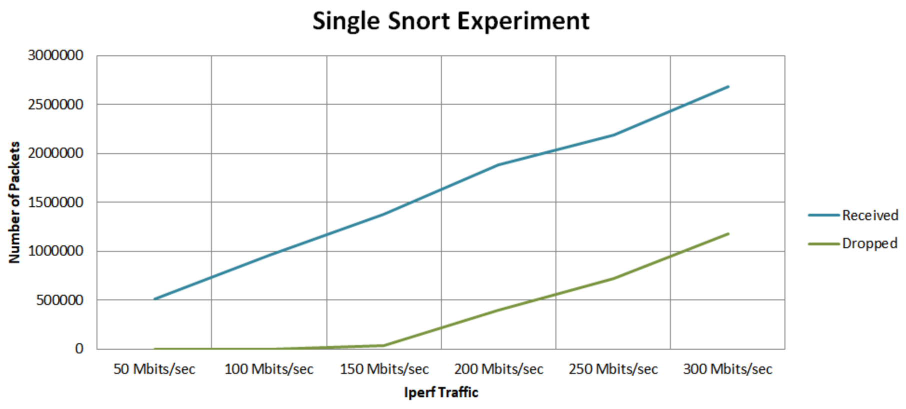


Figure 27: Packets received and dropped for Single Snort IDS

A single snort instance at a 50 % CPU utilization limit was used in this experiment. An Iperf instance was used to simulate network traffic and generated UDP packets incrementally by 50 mbits/second. During the traffic generation of Iperf, a Skipfish instance is initialized to simulate a DoS attack where the Snort instance generates alerts. This experiment was run to see at what point Snort drops packets and when to add another Snort instance. As seen in Figure 27, packets begin to drop when the traffic passes 150 Mbits/Sec. At 150 Mbits/sec, 2.67% of all the packets received were dropped. At this point, it does not affect the IDS ability to detect attacks. At 200 Mbits/Sec, there is a 21.29 % drop in packets. This is when the IDS performance becomes a serious issue. At 21.29 %, the IDS can let a significant amount of malicious traffic through without any detection. At this point, it is important to consider adding more sensors and distributing the workload. At 250 Mbits/Sec, 33.10 % of the packets are dropped and at 300 Mbits/sec, 44.81 % of the packets are dropped. By this point, a significant majority of packets are being dropped which results in losing valuable information such as suspicious traffic and requests.

6.1.2 Experiment Two: Two Snort IDS

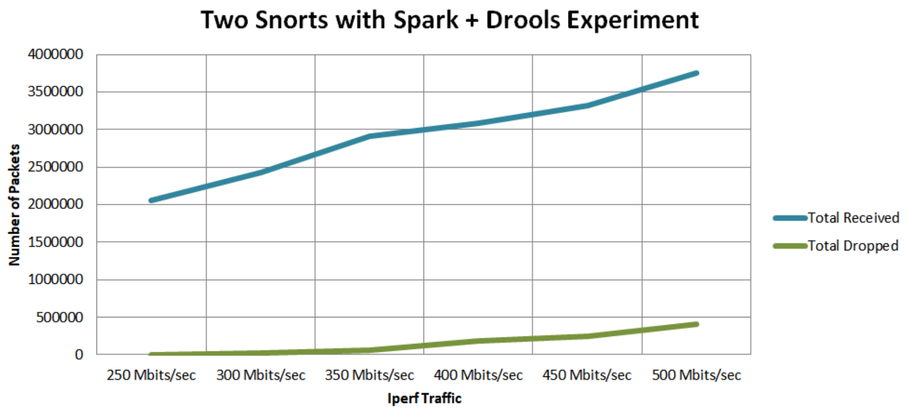


Figure 28: Packets received and dropped for Two Snort IDS

After observing the issues with having a single mechanism for detecting attacks, we added two Snort IDS sensors. These two snort instances were each running at 50 % CPU utilization limit for this experiment. As seen in Figure 28, there was no packets being dropped when the traffic was under 250 Mbits/sec. This was already a significant improvement, as previously with a single sensor, we would be seeing a 30 % drop at this point. At 300 Mbits/sec, there are only 0.97 % packets being dropped. This is an insignificant amount of dropped packets and suspicious traffic is still detected at this point. At 350 mbits/sec, 2.02 % of packets are being dropped. We start seeing more packets dropped at the 450 mbits/sec and 500 mbits/sec. At 450 mbits/sec, 7.5 % of all packets are dropped and at 500 mbits/sec, 10.78 % of all packets are dropped. As demonstrated, adding more sensors begin to decrease the number of packets dropped. However, when we begin distributing the sensors, 50 % of all the packets are transferred to one Snort IDS, and the other 50 % to the second sensor. With our architecture, we can take advantage of distributed sensors and send the individual sensor alerts to our architecture to perform an aggregated analysis.

6.1.3 Experiment Three: Three Snort IDS

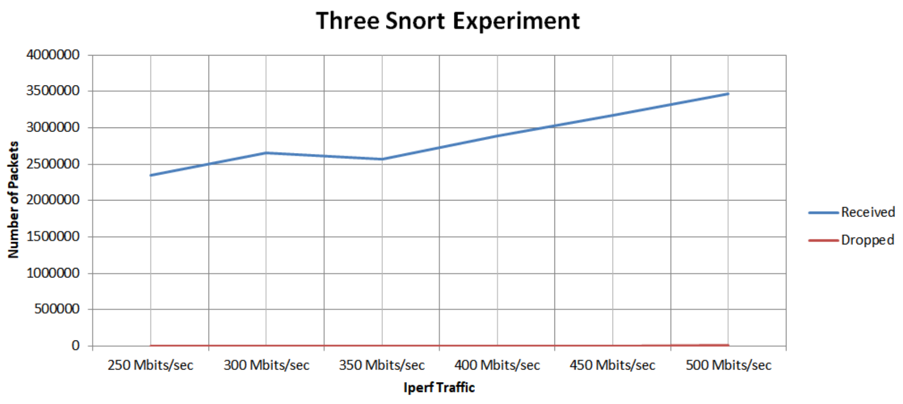
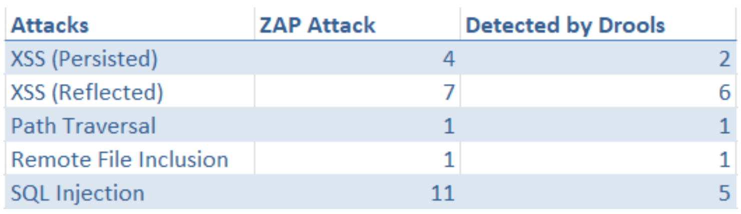


Figure 29: Packets received and dropped for Three Snort IDS

In this experiment, we look at three snort instances, each running at 50 % CPU utilization limit. Compared to the previous experiment, there is little to none packet drop throughout the workloads as seen in Figure 29. At 500 mbits/sec, there is only a 0.23 % drop in all packets, an insignificant amount for the highest traffic load of the experiments. This will result in capturing the entirety of the incoming traffic and any indication of a potential attack will not be lost due to non-distributed IDS limitation. We can then feed this data into our architecture to combine the information from the distributed IDS to understand the complete picture of our system.

6.2 Vulnerability Table detected by Drools

The key part of our platform is using the written business rules to detect security threats of an application. In this section, we present our results of the application-specific use case from Chapter 4. We demonstrate our platform and demonstrate the ability of business rules to detect these security threats by using the ZAP active scanner and manually injecting the found vulnerabilities. We then created a vulnerability table comparing the threats that ZAP detected with the amount of threats our platform was able to detect.



Attacks	ZAP Attack	Detected by Drools
XSS (Persisted)	4	2
XSS (Reflected)	7	6
Path Traversal	1	1
Remote File Inclusion	1	1
SQL Injection	11	5

Figure 30: Vulnerability Table for ZAP results

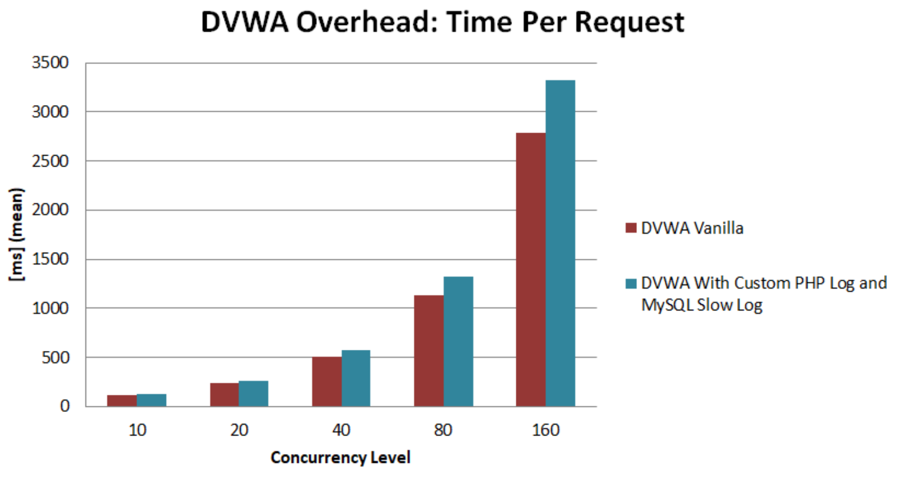
The vulnerability table can be seen in Figure 30, where the ZAP Scanner detected vulnerabilities in the Damn web application. A total of 11 XSS, 1 Path Traversal, 1 RFI and 11 SQL Injection vulnerabilities were detected in the web application. We used ZAP to then manually inject the attacks while our platform was running and protecting the application. The results of the detected threats is shown in Figure 30. From the figure, our platform detected a total of 8 XSS attacks, 1 Path Traversal Attack, 1 RFI and 1 SQL Injection.

However, our platform was able to detect only 62.5 % of the ZAP attacks on the application. However, on further inspection, 37.5 % of the vulnerabilities were false positives that ZAP considered vulnerabilities. In the XSS case, ZAP would leave a script on its initial attack, but when revisiting the page and using the GET response, ZAP thinks its previous script is a new XSS attack. However, our platform would detect this XSS attack when it was originally inserted into the page. The 3 XSS attacks that Drools did not detect are GET responses that did not send a script. There were 11 SQL Injections found by ZAP but 6 of the attacks were false positives. One of the false SQL Injection was a regular login and the other 5 never queried the SQL database. When not including the false positives, Drools was able to detect all the application vulnerabilities that ZAP can exploit. This experiment shows that it is possible to use business rules to detect a wide variety of security threats.

6.3 Web Performance for End-User

In this section, we begin to look at our platform's effect on the performance of the web application. We were interested in seeing the overhead that may arise when implementing the components that are required of the platform such as the logs, configurations and the network utility tools. We use the same setup as the Vulnerability Table experiment, with one DVWA web application and our platform which was hosted on a seperate machine. We then use Apache Bench, a benchmarking tool that sends traffic to the web application. Apache Bench reports the Requests per Second, the Time per Request and the Transfer Rate. The web application was implemented as a docker container. Each experiment was run 10 times each, with 5000 number of requests. For each iteration, we increase the number of concurrent users against the web application. The web application, Apache Bench and the Platform were hosted on virtual machines each with 4GB RAM, 2 VCPUs and a 40 GB Disk.

In the first experiment shown in Figure 31 and Figure 32, Netcat was installed on the DVWA web application to stream the MySQL Slow Log and a Custom PHP log to the Platform. This set-up of the DVWA web application is required for our Platform. We then compared it to a Vanilla web application without any logging or security protection, and observed the overhead. When using our custom application over the Vanilla application, the highest overhead for Time per Request is 18.93 % for 160 Concurrent users, the lowest is 11.50 % for 20 concurrent users and the average is is 14.40 %, with the custom DVWA web application taking more time to complete the request. For Requests per Second, the highest overhead is 15.88 % for 160 Concurrent

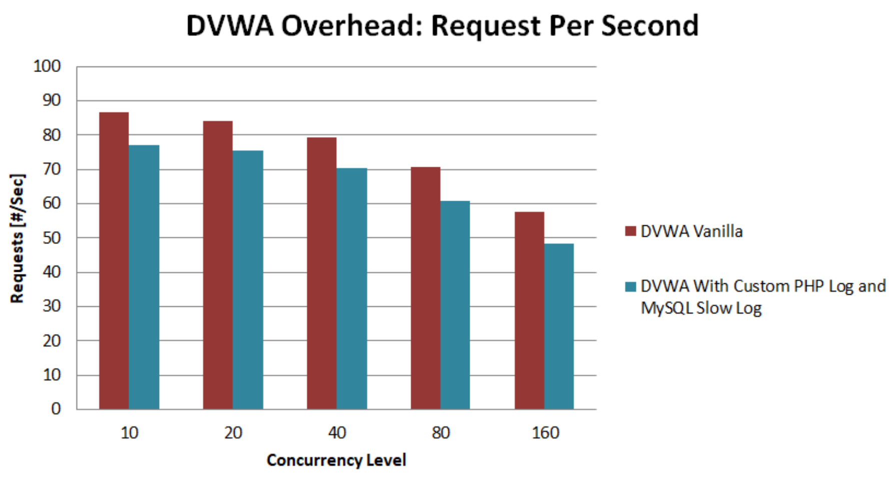


(a) Time Per Request

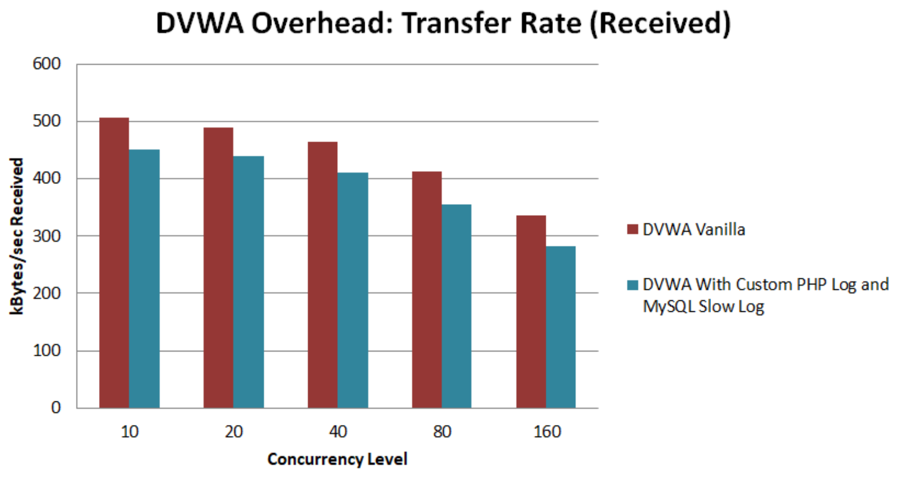
Figure 31: DVWA End-User Time Per Request Performance Overhead

users, the lowest is 10.21 % for 20 concurrent users and the average is 12.52 %, with the custom DVWA web application handling less requests per second. For the transfer rate, the highest overhead is 15.88 % for 160 concurrent users, the lowest overhead is 10.30 % for 20 concurrent users, and the average overhead for all the concurrent users is 12.52 % with the custom DVWA web application performing slower.

These first experiments show that there is a overhead when implementing the components for our platform. We looked at isolating the specific components to find what may be causing this overhead. Upon further inspection, we found that Netcat was causing the majority of the overhead to the DVWA web application. To reduce the Netcat overhead, we created a separate Secure Shell File System (SSHFS) container where the log data was to be mounted. SSHFS was used to mount the log data in the DVWA container to this seperate container, where Netcat was running. This shift was able to fix the overhead significantly. However, one of the drawbacks is that the rate of the mounted data that is fed to the Drools engine is about 5 to 10 seconds slower. This

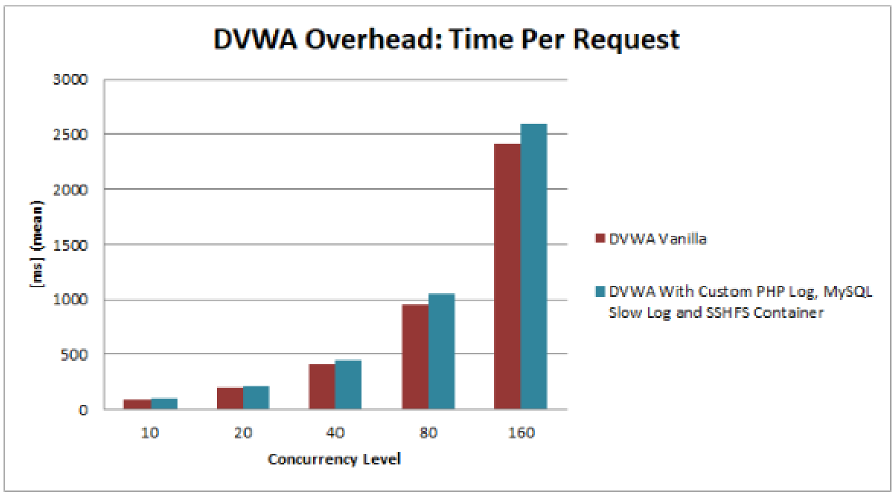


(a) Request Per Second



(b) Transfer Rate

Figure 32: DVWA End-User Request and Transfer Performance Overhead

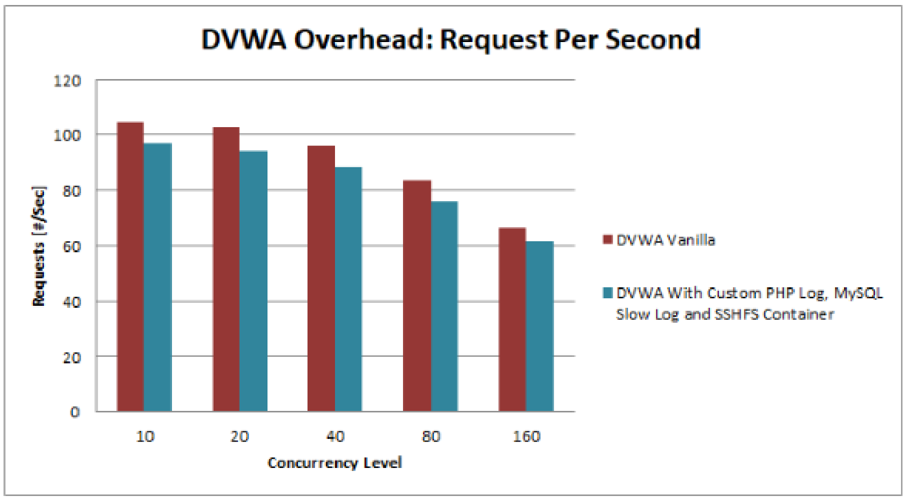


(a) Time Per Request

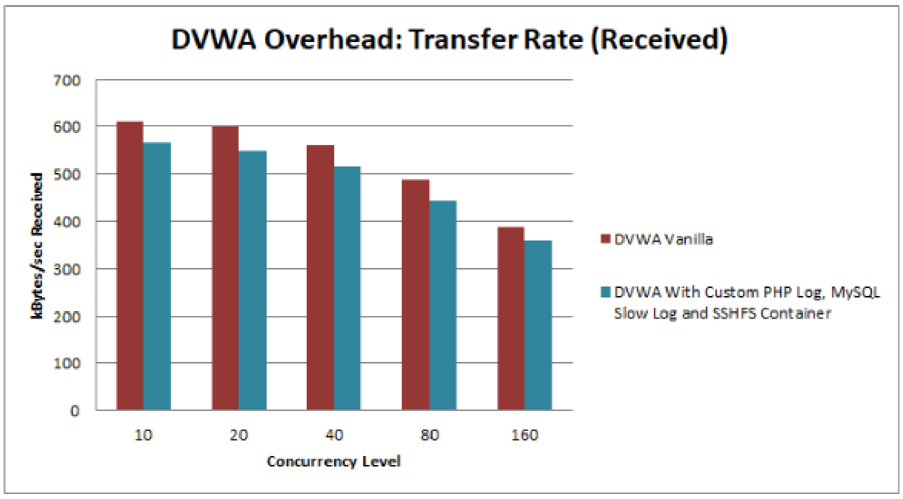
Figure 33: DVWA End-User Time Per Request Performance Overhead

issue has to be explored in the future to find additional solutions. Another solution to reduce overhead was moving all the log data into the RAM memory.

The results of moving Netcat to a seperate container and moving all the logs to the RAM can be seen in our experiments shown in Figure 33 and Figure 34. From the figures, we show that there is a significantly reduced overhead between Vanilla and the Protected web application. For Time per Request, the highest overhead is 10.55 % for 80 Concurrent users, the lowest is 7.76 % for 10 concurrent users and the average is 8.75 %, with the custom DVWA web application taking more time to complete the request. For Requests per Second, the highest overhead is 9.56 % for 80 Concurrent users, the lowest is 7.13 % for 10 concurrent users and the average is 8.05 %, with the custom DVWA web application handling less requests per second. For the transfer rate, the highest overhead is 9.56 % for 80 concurrent users, the lowest is 7.13 % for 10 concurrent users, and the average overhead for all concurrent users is 8.04 % with the custom DVWA web application performing slower. Although there is still an



(a) Request Per Second



(b) Transfer Rate

Figure 34: DVWA End-User Request and Transfer Performance Overhead

overall present overhead of about 8 % for each of the experiments, this is minimal to the user impact and our architecture vastly improves on the security and monitoring of the vanilla application.

6.4 Summary

In this chapter, we report our findings for three main experiments related to our platform. We ran an experiment on distributing IDS sensors, detecting vulnerabilities and looking at the performance overhead. For the first experiments, we generated traffic load on one sensor, two sensors and three sensors, and observed the packet droppage. By distributing the sensors, we were able to show a minimize the packet droppage significantly in our results. The second experiment demonstrated our platform and it's ability to detect vulnerabilities. Using manually injected attacks from the vulnerabilities found through the ZAP active scanner, we presented a table showing the list of vulnerabilities and the attacks our platform was able to detect. We also discussed the false-positives in this experiment. Our final experiment was to observe the performance effects for the end user when implementing our platform with the web application. Using a benchmarking tool, we found that there was an overhead for Requests per Second, Time per Request and the Transfer Rate.

Chapter 7

Conclusion

Vulnerabilities in applications are a constant security threat. These vulnerabilities can be exploited and result in serious breaches and malicious attacks. However, these vulnerabilities are often the result of the barriers of communication between developers and security analysts. By providing a collaborative solution to bridge the communication gap, we can create a stronger and more secure system. Additionally, there are a multitude of data sources that can be analyzed, each generating a number of security threats. Analyzing one sensor or data source out of many does not provide us with the complete state of the system. These multiple data sources also produce a large and growing ruleset. Distributing and maintaining the ruleset, aggregating the data from multiple sources and performing sophisticated analysis is required for a security control system.

In this work, we designed a distributed and hierarchical Rule-based Security Management architecture to support SecDevOps. This collaborative solution allows the developers and analysts to define security rules during runtime at an application level and to use a wide variety of different data sources showing our architecture's ability to be flexible and expandable. The architecture can handle high volumes of workloads and aggregate data from different sources to effectively monitor and understand the complete state of the system, allowing us to be aware of more sophisticated attacks. In addition, the architecture is capable of executing adaptive actions to mitigate security threats.

For our first contribution, we designed an architecture that uses the BRE with a Streaming Analytics Cluster that can analyze incoming data and run them against customizable business rules. This creates a collaborative platform for developers and security teams, providing a simplified way of defining security rules at the operational level. This architecture allows for distributing the workers and streaming clusters throughout the network and for distributing a large business ruleset among the clusters. We distributed the architecture as a hierarchy, with each cluster responsible for its own ruleset and higher level clusters responsible for more sophisticated analysis.

For our second contribution, we closed the MAPE-K Loop with our architecture by designing several adaptive functions for the platform. These adaptive actions helped us to write rules that can not only detect vulnerabilities, but also mitigate these attacks and security threats.

For our third contribution, we implemented our platform with practical real life use cases. Our platform was used as a solution for security threats that affect these cases, and demonstrated its ability to be easily integrated with different types of applications and security threats.

For our fourth contribution, we tested our platform on an industrial production application, Bitnobi. By using Business Rules to specify security conditions and actions, we enabled two different teams to have a common understanding of the problems and solutions. The developers were able to write their own rule-based patches when they were aware of a vulnerability in the system. Our case study showed that we were able to detect vulnerabilities more effectively, execute adaptive actions to mitigate the vulnerabilities and notify the teams, thus creating a stronger secure web application. We were able to seamlessly keep the system in constant monitoring and maintenance while the developers continued working on their application.

For our final contribution, we designed three evaluation experiments for our architecture. The first experiment was to view the limitation of an IDS, as we evaluated a single sensor versus distributed sensors to observe packet loss from incoming traffic. Our second experiment was to demonstrate the platform's ability to detect application-specific vulnerabilities and attacks by manually injecting vulnerabilities our active scanner found. We also evaluated the performance overhead of our platform to the end-user, and provided technical solutions to decrease the overhead.

7.1 Future Works

While we designed a collaborative platform and evaluated the ability to detect security threats, we need to examine an in-depth analysis on the scalability of our proposed solution. We plan on evaluating our architecture's scalability by designing a validation experiment that examines how many security events our architecture can handle at a time, and how our architecture will handle a growing number of security events. We will also look at effects of scaling the clusters and worker nodes with the number of incoming security events. This will be the next key step to evaluate our architecture.

We also aim to add more adaptive actions to our architecture. We plan on looking at more security use cases and what adaptive actions may benefit them. Many web applications include REST APIs, and creating an adaptive action that can make REST calls can be a general benefit to multiple security and development cases. Additionally, we plan on creating an adaptive action that can modify the Docker platform by adding, removing or updating containers. This can allow the developers or security teams to make system-wide adaptive changes such as adding new sensors or adding a new containerized web application.

Finally, we plan on trying to implement our platform with more complex scenarios in the future, as skilled attackers can be slow and methodical and their preparation for an attack may take months [50]. Using the stateful and temporal properties of Business Rules, we will explore detecting prolonged suspicious behavior in a larger time frame.

Bibliography

[1] R. Vaclav, *Software engineering: the current practice*. Chapman and Hall/CRC, 2012.

[2] C. Wysopal, L. Nelson, E. Dustin, and D. Dai Zovi, *The art of software security testing identifying software security flaws*. Addison-Wesley, 2007.

[3] *Ponemon Institute's 2017 Cost of Data Breach Study: Global Overview*. [Online]. Available: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEL03130WWEN

[4] R. E. Smith, *Elementary information security*. Jones & Bartlett Learning, 2016.

[5] *Data Breach Aftermath and Recovery for Individuals and Institution proceedings of a workshop*. National Academies of Sciences, Engineering, and Medicine, 2016.

[6] M. Hüttermann, *DevOps for developers*. Apress, 2012.

[7] V. Mohan and L. B. Othmane, "Secdevops: Is it a marketing buzzword? - mapping research on security in devops," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, Aug 2016, pp. 542–547.

[8] J. R. Vacca and J. R. Vacca, *Computer and Information Security Handbook, Second Edition*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[9] J. Boyer and H. Mili, *Agile Business Rule Development Process, Architecture, and JRules Examples*. Springer Berlin, 2014.

[10] D. Hay, K. Healy, J. Hall, C. Bachman, J. Breal, J. Funk, J. Healy, D. McBride, R. McKee, and T. Moriarty, "Defining business rules. what are they really? the business rules group," pp. 4–5, 01 2000.

[11] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland, "Characterizing the performance of network intrusion detection sensors," in *Recent Advances in Intrusion Detection*, G. Vigna, C. Kruegel, and E. Jonsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 155–172.

[12] P. Mell and T. Grance, *The NIST definition of cloud computing*. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011.

[13] A. Bahga and V. K. Madisetti, *Cloud computing: a hands-on approach*. CreateSpace Independent Publishing Platform, 2013.

[14] A. V. Vasilakos, R. Beraldi, R. Friedman, and M. Mamei, *Autonomic Computing and Communications Systems: Third International ICST Conference, Autonomics 2009, Limassol, Cyprus, September 9-11, 2009, Revised Selected Papers*. Springer, 2010.

[15] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, May 2015, pp. 13–23.

[16] B. Wilder, *Cloud architecture patterns*. OReilly, 2012.

[17] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice Architecture: Aligning Principles, Practices, And Culture*. OReilly, 2016.

[18] A. Mouat, *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media, Inc.

[19] S. Sharma, R. RV, and D. Gonzalez, *Microservices: Building Scalable Software*. Packt Publishing Ltd, 2017.

[20] N. Dasgupta, *Practical Big Data Analytics: Hands-On Techniques to Implement Enterprise Analytics and Machine Learning using Hadoop, Spark, NoSQL and R*. Packt Publishing, 2018.

[21] M. Guller, *Big Data Analytics with Spark: A Practitioners Guide to using Spark for Large-scale Data Processing, Machine Learning, and Graph Analytics, and High-Velocity Data Stream Processing*. Apress, 2015.

[22] "Analytics." [Online]. Available: https://www.ibm.com/analytics/hadoop/mapreduce

[23] "Spark streaming programming guide." [Online]. Available: https://spark.apache.org/docs/latest/streaming-programming-guide.html

[24] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228301

[25] J. Davis and K. Daniels, *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. OReilly, 2016.

[26] S. Vadapalli, *DevOps: Continuous Delivery, Integration, and Deployment with DevOps*. Packt Publishing Ltd, 2018.

[27] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer, "Stateful intrusion detection for high-speed networks," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, ser. SP '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 285–. [Online]. Available: http://dl.acm.org/citation.cfm?id=829514.830527

[28] M.-Y. Huang, R. J. Jasper, and T. M. Wicks, "A large scale distributed intrusion detection framework based on attack strategy analysis," *Computer Networks*, vol. 31, no. 23, pp. 2465 – 2475, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128699001140

[29] V. Sekar, R. Krishnaswamy, A. Gupta, and M. K. Reiter, "Network-wide deployment of intrusion detection and prevention systems," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 18.

[30] O. B. Fredj, H. Sallay, A. Ammar, M. Rouached, K. Al-Shalfan, and M. B. Saad, "On distributed intrusion detection systems design for high speed networks," in *Proceedings of the 9th WSEAS International Conference on Advances in E-Activities, Information Security and Privacy,(ISPACT'10), WSEAS, USA*, 2010, pp. 115–120.

[31] G. Vigna, R. A. Kemmerer, and P. Blix, "Designing a web of highly-configurable intrusion detection sensors," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, ser. RAID '00. London, UK, UK: Springer-Verlag, 2001, pp. 69–84. [Online]. Available: http://dl.acm.org/citation.cfm?id=645839.670737

[32] V. Ramsurrun and K. Soyjaudah, "The stateful cluster security gateway (csg) architecture for robust switched linux cluster security," in *Proceedings of the Seventh Australasian Conference on Information Security-Volume 98*. Australian Computer Society, Inc., 2009, pp. 109–118.

[33] C. Ko, D. A. Frincke, T. Goan, Jr., T. Heberlein, K. Levitt, B. Mukherjee, and C. Wee, "Analysis of an algorithm for distributed recognition and accountability," in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, ser. CCS '93. New York, NY, USA: ACM, 1993, pp. 154–164. [Online]. Available: http://doi.acm.org/10.1145/168588.168608

[34] T. Lojka, P. Papcun, and I. Zolotová, "Business rule engine for education in virtual laboratory cyberlabtrainsystem," in *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Jan 2017, pp. 000 343–000 346.

[35] C. Nagl, F. Rosenberg, and S. Dustdar, "Vidre–a distributed service-oriented business rule engine based on ruleml," in *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, Oct 2006, pp. 35–44.

[36] X. Feng and M. Subramanian, "Incorporating business rule engine technology in control center applications-toward adaptive it solutions," in *2008 IEEE Energy 2030 Conference*, Nov 2008, pp. 1–5.

[37] A. Chuvakin, K. Schmidt, and C. Phillips, *Logging and log management: the authoritative guide to understanding the concepts surrounding logging and log management*. Elsevier/Syngress, 2013.

[38] A. Buecker, J. Amado, D. Druker, C. Lorenz, F. Muehlenbrock, and R. Tan, *IT security compliance management design guide with IBM Tivoli Security Information and Event Manager*. IBM, International Technical Support Organization, 2010.

[39] F. Holik, J. Horalek, S. Neradova, S. Zitta, and O. Marik, "The deployment of security information and event management in cloud infrastructure," in *2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA)*, 2015, pp. 399–404.

[40] "Ibm qradar siem," Jul 2017. [Online]. Available: https://www.ibm.com/us-en/marketplace/ibm-qradar-siem/details

[41] 2018. [Online]. Available: https://docs.jboss.org/drools/release/7.7.0.Final/drools-docs/html_single/index.html

[42] D. P. Silberberg and G. E. Mitzel, "Information systems engineering," *Johns hopkins ApL TechnicAL DigesT,*, vol. 26, no. 4, p. 343–349, 2005.

[43] S. Casteleyn, F. Daniel, P. Dolog, and M. Matera, *Engineering Web Applications*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[44] S. Jacobs, *Engineering Information Security: The Application of Systems Engineering Concepts to Achieve Information Assurance*, 2nd ed. Wiley-IEEE Press, 2015.

[45] "Docker overview," Jul 2017. [Online]. Available: https://docs.docker.com/engine/docker-overview/

[46] "Snort: The world's most widely deployed ips technology," Jan 2015. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/security/brief_c17-733286.html

[47] M. T. Chung, N. Quang-Hung, M. T. Nguyen, and N. Thoai, "Using docker in high performance computing applications," in *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, July 2016, pp. 52–57.

[48] Y. Chung, "Distributed denial of service is a scalability problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 1, pp. 69–71, 2012.

[49] "Open web application security project," Jun 2018. [Online]. Available: https://www.owasp.org/index.php/

[50] J. S. Chris Sanders, *Applied Network Security Monitoring: Collection, Detection, and Analysis*. Syngress, 2013.