RULES BASED ANALYSIS ENGINE FOR APPLICATION LAYER IDS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

David Scrobonia

March 2017

COMMITTEE MEMBERSHIP

TITLE:	Rules Based Analysis Engine for Application Layer IDS
AUTHOR:	David Scrobonia
DATE SUBMITTED:	March 2017
COMMITTEE CHAIR:	Zachary Peterson, Ph.D. Department of Computer Science
COMMITTEE MEMBER:	Bruce DeBruhl, Ph.D. Department of Computer Science
COMMITTEE MEMBER:	John Seng, Ph.D. Department of Computer Science

ABSTRACT

Rules Based Analysis Engine for Application Layer IDS

David Scrobonia

Web application attack volume, complexity, and costs have risen as people, companies, and entire industries move online. Solutions implemented to defend web applications against malicious activity have traditionally been implemented at the network or host layer. While this is helpful for detecting some attacks, it does not provide the granularity to see malicious behavior occurring at the application layer. The AppSensor project, an application level intrusion detection system (IDS), is an example of a tool that operates in this layer. AppSensor monitors users within the application by observing activity in suspicious areas not able to be seen by traditional network layer tools. This thesis aims to improve the state of web application security by supporting the development of the AppSensor project. Specifically, this thesis entails contributing a rules-based analysis engine to provide a new method for determining whether suspicious activity constitutes an attack. The rules-based method aggregates information from multiple sources into a logical rule to identify malicious activity, as opposed to relying on a single source of information. The rules-based analysis engine is designed to offer more flexible configuration for administrators and more accurate results than the incumbent analysis engine. Tests indicate that the new engine should not hamper the performance of AppSensor and use cases highlight how rules can be leveraged for more accurate results.

ACKNOWLEDGMENTS

Thanks to John Melton for guiding me through the AppSensor Project and supporting my contributions.

Thanks to Dick Hartung for support and mentorship throughout my collegiate life.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Attacks on web applications are becoming more frequent, more complex, and more potent. [13] Attackers are using more advanced methods, more complex combinations of techniques, and expanding their area of attack. Additionally, web applications are becoming larger and more complex. [29] There are a greater number of web applications that are storing credit card information, passwords, and personal identifiable information (PII), increasing the cost of being breached. [16] The larger they become the larger the attack surface becomes, and the harder it becomes to properly maintain and monitor all features of an application. Many administrators have security tools such as firewalls, WAF's, and vulnerability scanners. [26] While these tools can view network traffic and identify attacks at a crude, broad level, they are only able to detect the obvious and apparent.

Administrators do not always have the ability to view their applications at a finer grained level. They are unable to see exactly where in the web application code an attack is targeting and can not necessarily determine if an attacker is probing their site for vulnerabilities. Attackers are easily able to probe websites for vulnerabilities without being detected and when they are able to exploit an application it's not easy to determine where and how they were able to breach the application.

One solution to this problem is to use an application level intrusion detection system (IDS). An application level IDS embeds sensors within the code of a web application to relay information about malicious activity to an administrator. From there, administrators are able to see what kind of malicious activity is occurring in what area of the application, right down to the form or field. One example of an application level IDS is the open source AppSensor project from OWASP. [38]

The AppSensor project supports real time detection and automated response of malicious activity at the application layer. The AppSensor system runs as a standalone service that integrates with a web application through a system of embedded sensors. These sensors are in reality just a few lines of code, included at points of interest in the web application, that communicate to the AppSensor server that a specific user has done a specific action.

For example, a sensor may be included in the authentication component of the web application where a failed login is handled. Perhaps there is a check on the string length of the username entered, and if a user were to attempt to enter more than the allowed amount of characters this could represent malicious activity. [8]

```
if ( username.length > 30 )
{
    # ``AE1'' is the identifier for this specific sensor
    appSensor.addEvent ( ip_address, ``AE1'' )
}
```

When a user fails to login it would send an *event* of type "AE1" to the AppSensor server where it is processed by several analysis engines.

While the AppSensor project is a mature project, there is still need for further development. One area of needed development is in the analysis engines. The analysis engines determine whether the input from sensors constitutes an attack and then handles the resulting actions. AppSensor was designed to support multiple, interchangeable analysis engines, but prior to this project only offered one analysis option. The lone option's means of detection is to check incoming events against predefined thresholds. A *threshold* defines a number of events that occur in a period of time, so that if X number of events happen in Y seconds/minutes/etc, then an attack should be triggered. Using the example from before, a threshold could be defined so that if a

user fails the username check more than 20 times in one minute, the analysis engines would determine this to be an attack.

```
threshold_1:


if (AE1 occurs > 20 in 1 minute)
{
    appSensor.triggerAttack(``threshold_1'', event.user)
}
```

While this method is useful in its simplicity and robustness, it has it's drawbacks. For one, it limits administrators to using a single heuristic for determining an attack because the threshold model only evaluates a single type of sensor at a time. Ideally administrators would have more control over using multiple sources of information to create more complex, higher-level detection methods. Secondly, because only one heuristic is evaluated at a time a threshold's number of events must be defined relatively high to avoid false positives, which in turn may lead to more false negatives. This property of the threshold model leaves administrators with the problem of choosing to try to prevent false negatives or false positives, with little wiggle room.

For my thesis I created a new set of "rules-based" analysis engines to improve upon these flaws and offer users of AppSensor an alternative method of analysis. The rules-based approach aggregates multiple thresholds into a rule using the logical operators AND and OR as well as the temporal operator THEN.

From the previous example of the failed login, an administrator may not want to depend solely on a username character length check to trigger an attack, but rather include several different failed login heuristics. They may want to also check whether illegal characters were used (defined as threshold_2), if unusual headers were included

(defined as threshold_3), or the total number of valid and invalid login attempts was over a certain amount (defined as threshold_4). They could then craft a rule to include all of these elements.

```
rule_1:

if (threshold_1 OR threshold_2 OR threshold_3 AND threshold_4)
{
    appSensor.triggerAttack(``rule_1'', event.user)
}
```

This not only provides administrators more control and creativity in how they would like to define their rules for detection, but also allows for improved accuracy by leveraging multiple heuristics. False negatives can be controlled by lowering the number of events in a threshold, and false positives can be controlled by adding more than one heuristic. The rules-based analysis engines can replace the original engines or can be used along side of.

The rules-based analysis engine has been supported with unit tests, integration tests, and performance tests. Performance tests show that while the rules-based analysis engine is slower than the incumbent reference engine, processing time is still within the reasonable limits of operation. Currently, the rules-based analysis engine only integrates with the base configuration of AppSensor, but work is already being conducted to fully integrate the engine with all AppSensor components.

The rules-based analysis engine was added to the official AppSensor project in January 2017.

Chapter 2

BACKGROUND

2.1 Web Application Basics

While providing a full background of the structure, history, and development of web applications would be out of the scope of this thesis, I feel it is important to highlight certain web application *basics* so that the reader can fully understand this work. [29] provides a very efficient and thorough overview which I will lean heavily on here to provide necessary background. I urge readers to reference this survey if they have further questions.

2.1.1 The History and Growth of Web Insecurity

In the rush to develop the web from a small network of static pages to the behemoth of dynamic and distributed web applications that power the world today, strong security features were often overlooked. Applications delivering services with sensitive and valuable data became commonplace quicker than security development could keep pace with. These applications provided a large target to attack, and a financially lucrative target at that. Black markets provide a marketplace to sell many kinds of personally identifiable information (PII) including passwords, social security numbers, and credit card numbers.

The sheer number of technologies used to power the web and web applications introduced a level of complexity in the ecosystem that created massive challenges for developing proper security. Looking back at the early technologies of web applications through a retrospective lens it is easy to understand how security problems arose.

The foundation of client technology was built on Javascript and Flash. Javascript,

still the client side technology of choice today, is a notoriously quirky language due to its infamous origins of being written in 10 days. [10] These quirks often lead to inconsistent and difficult to understand code - a breeding ground for vulnerabilities. Javascript is also a powerful language that is tightly woven into the browser, a combination that can lead to cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks that will be discussed later. Javascript has been able to evolve into a popular and useable modern language, however, with the help of well developed frameworks. Flash on the other hand, still remains a much maligned browser plugin due to an extremely buggy environment that leads constant exploitation. [39] [24] Foundational server side technologies like ASP and PHP did not offer many built in security features to support things like secure authentication, authorization, or encryption.

The goal is not to bash these early languages and frameworks, as many elements of the web application security ecosystem were still being researched and developed. Rather the goal is to highlight that the world of web applications was built on technologies that were not often designed with security in mind, and the consequences of that can still be seen today. Developing modern secure web applications is a non-trivial problem, that requires a serious focus to address increasing attacks.

2.1.2 Application Security Basics

There are several key concepts and definitions that are important to understand when discussing web application security, so before progressing I want to be sure these concepts are properly defined and explained.

A *vulnerability* in a web application is "a hole or a weakness in the application, which can be a design flaw or an implementation bug, that allows an attacker to cause harm to the stakeholders of an application." [18] A web application attacker

will search a site for a vulnerability so that they may *exploit* it. *Exploitation* is the process of leveraging a vulnerability to cause harm or damage. For example, an application may have a vulnerability where poorly implemented SQL database allows any user to run arbitrary SQL commands. An attacker could exploit this vulnerability to query for sensitive data in the database. Used as a noun, an exploit refers to a malicious script or code designed to exploit a vulnerability. An *attack* is an attempt to exploit a web application regardless of whether it was successful or not. Many of these terms are used interchangeably in the practice, but I will attempt to stick to these definitions for clarity.

There are many classifications of the types of web application vulnerabilities. For brevity I will discuss several key vulnerabilities taken from the OWASP Top 10. The *OWASP Top 10* is a list of the consensus 10 most critical web vulnerabilities found in web applications. [11]

One of the most common web vulnerabilities is SQL injection. A *SQL injection* (SQLi) vulnerability occurs when a web application executes a SQL query with untrusted user data. Exploitations of SQLi vulnerability can lead to stolen data, deleted data, or even escalation of privileges and remote command execution (RCE). SQLi vulnerabilities have been significantly reduced in recent years due to increased awareness and the proper use of prepared statements and parameterized queries.

Cross-site scripting (XSS) is a vulnerability that allows malicious scripts to run on an otherwise benign web application. This occurs when the application does not properly validate and sanitize user input, and then echos the malicious script back to a victim client, who incidentally executes the script. The most common way this occurs is through Javascript being maliciously included in forum posts or comments and being executed by a victim who then views the forum or comment. XSS is the most prevalent web vulnerability, mainly due to the large surface that XSS can be

found in. XSS is traditionally used to steal a victim's session token, but can be used for installing other malicious files, stealing information, redirects, or anything else that can be done with Javascript.

Broken authentication and session management are a group of vulnerabilities found in poorly designed authentication and session management implementations. Because both areas are difficult to design and implement correctly, they are often prone to have vulnerabilities and to be heavily probed by attackers. Authentication provides a web application a means of confirming who a user is who they claim to be, while session management allows the web app to track a user's activity and keep them logged in over the stateless http protocol. Common vulnerabilities in authentication include poor password requirements, no lockout, information leak. Common vulnerabilities in session management include poor crypto, session-jacking, information leak.

The final class of vulnerabilities I will highlight are performed by *bypassing business logic*. All applications have logic specific to their own utility and often vulnerabilities can be exposed by circumventing the intended flow of this logic. As an example, consider an e-commerce application that has a multi-page checkout process. The first page lets you choose the product, the second page gathers payment information, and the last page is confirmation. An attacker may attempt to skip the second page and checkout without entering payment information. What will happen? Perhaps nothing malicious, or perhaps the adversary is able to order the product without paying. Depends on how the application was designed. Often an application's developer may not have considered this use case and a poorly designed application may not be prepared to securely handle it. There are many examples where business logic may be bypassed and lead to exploitation.

2.2 Increase in Web Application Attacks

Key industry reports presented in the last few years have indicated significant growth in the number of web application attacks. [13] [17] [14] [9]

The Web Application Attack Report (WAAR) is a report on web application security by Imperva which has been conducted the last 6 years with the most recent released in 2015. Imperva uses data received from their Imperva Web Application Firewall (WAF) product, implemented across 198 applications. [13] Imperva's Application Defense Center analyzed over 297,954 attacks from 22,850,023 captured security events, where an attack is defined a burst of related malicious events. The average SQL injection attack was built up of 72 malicious events.

Imperva concluded that the rate of attacks on web applications was increasing. Between 2014 and 2015 the number of SQL injection attacks increased by a factor of 3 and that cross-site scripting (XSS) attacks increased 2.5 times. Both attacks have been increasing the last two years. Additionally, 75% of the applications in their data set received each of the 8 attacks highlighted in their report.

Imperva also concluded that the intensity of attacks was increasing. They defined a "megatrend" when the ShellShock vulnerability went public and observed that 100% of their 198 monitored applications received an attack attempting to exploit the ShellShock vulnerability. This internet-wide blind scanning points to the mobility and aggression of web adversaries.

Verizon's Data Breach Investigation Report (DBIR) has been conducted since 2008 (with data since 200) and presents data from verified data breaches. [17] The most recent report released in 2016 further supports the growth in web application attacks, with an exponential increase in successful data breaches since the 2005. 27% of breaches at financial institutions involved web application attacks.

According to Alert Logic's 2015 Cloud Security Report, web application attacks on cloud applications have increased by 45% since 2014. [14] A 2011 survey found that 75% of responders had been "hacked" in the last 24 months. [9]

2.2.1 Cost

Calculating the risks and costs of successful web application attacks is a rough science at best. Work done at Ponemon Institute calculates the cost of data breaches using activity-based costing (ABC) which focuses on the cost of activities carried out by an organization as a result of a breach. [12] Although models to estimate risk and cost are currently being researched, early work is still able to portray a rough ballpark for the costs. In a 2011 survey half of IT practitioners estimated damages due to a web application attack would range between $200,000 - $500,000. [9]

While not all successful web application attacks will lead to a massive data breach, when they do the consequences are significant. A survey over 350 companies discovered that costs of a data breach have risen 23% since 2013, with the average data breach in 2015 costing the victim $3.79 million, as shown in Figure 2.1. [12] The majority of costs resulting from a data breach are spent on legal counsel and forensics, with a lesser amount spent on PR and credit monitoring for victims. [17] Some argue that the reputation damage due to a data breach also has a large impact on the cost of the attack with data indicating increasing loss of business. [12] Reputational damage leading to loss of business cost companies an average $1.57 million in 2015. Others argue that there is no long-term reputational damage by observing that the stock prices of both Home Depot and Target normalized, and ultimately increased, after their high profile breaches.

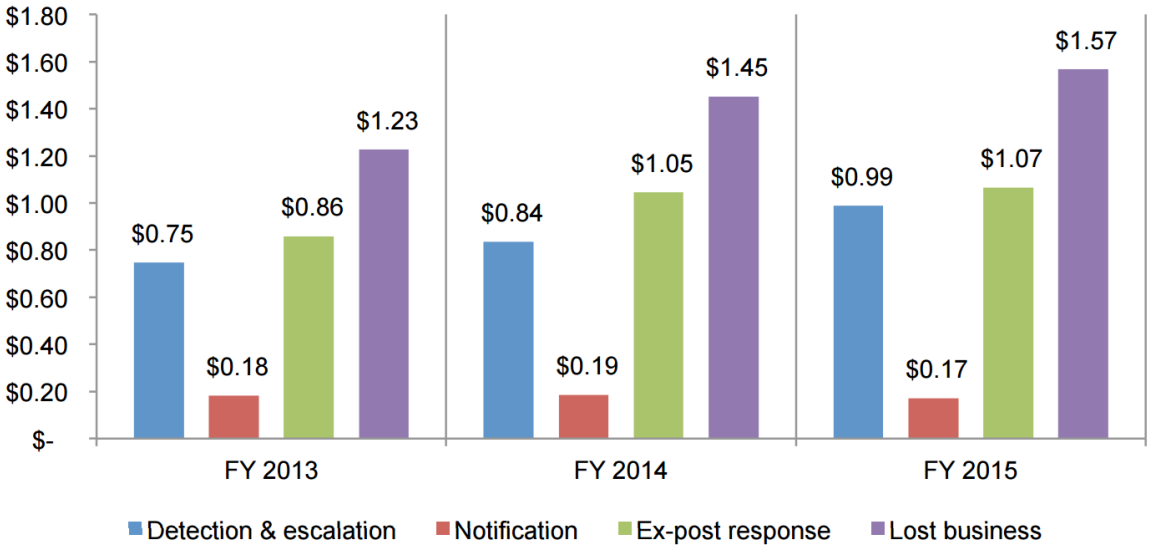


Figure 2.1: The increasing costs of web application breaches.

2.2.2 Targets

Companies from every industry are affected by web application attacks. Verizon's 2016 DBIR lists over 15 different distinct industries and highlights financial services and retailers as the key industries targeted by web application attacks. [17] Although Verizon concedes that the bulk of their web application data for 2016 emerged from a few sources, their conclusions can be validated by simply checking the news. In the last few years there have been high profile attacks on financial institutions like Chase [40], Qatar's National Bank [34] and retailers such as Home Depot and Target[27]. A survey conducted by LeaseWeb concludes that the industries accruing the highest costs from web application attacks are financial, public, and health services. [7] The high profile attack on Anthem health care shows not only is health care information a preferred target of attackers, but that the industry is a soft target. [32]

While all industries are affected, the cost to remediate a breach varies between industry. As can be seen in Figure 2.2, the healthcare industry was shown to have highest cost of a data breach, with each lost record costing them $363, nearly twice

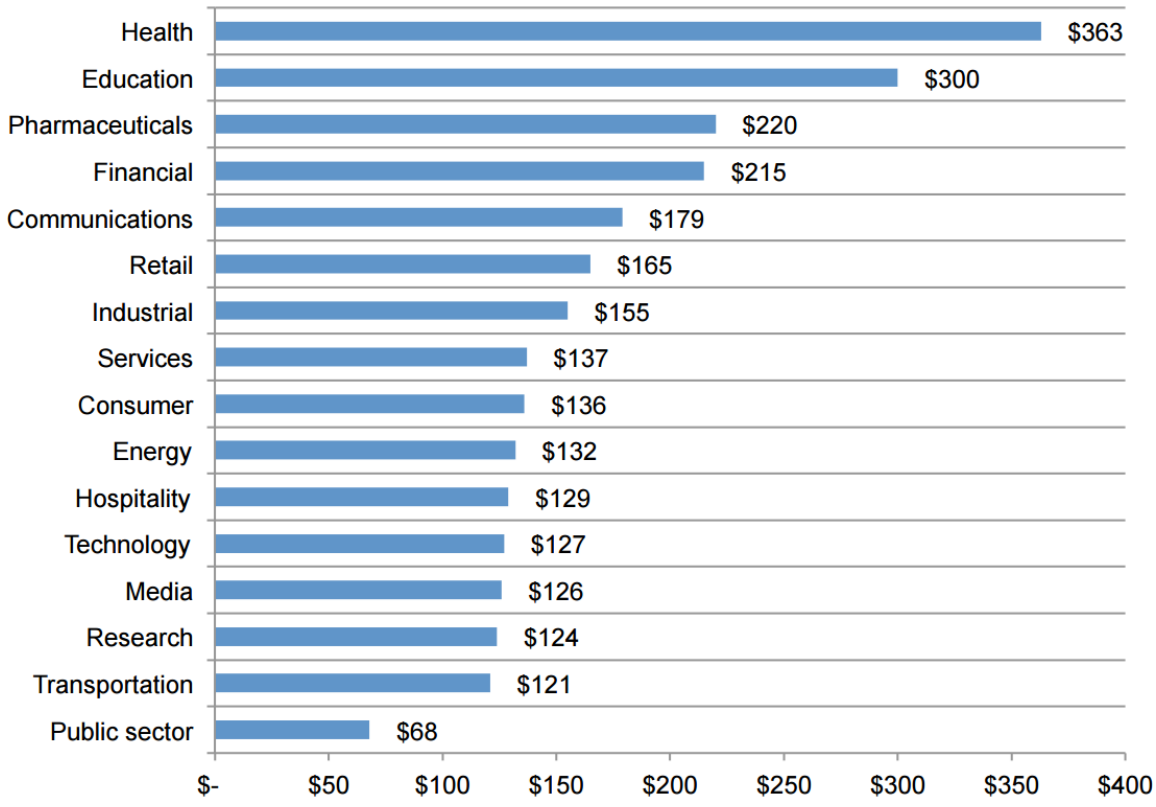


Figure 2.2: The cost of breach remediation by industry.

as much as the average of $154. [12]

2.3 Traditional Web Application Defenses

Web server architecture has often been viewed as an n-tiered system broken into different layers. One common configuration breaks the n-tiered architecture into 3 distinct layers: network, host, and application. [33] [28] In the context of security, each layer has a unique perspective on the web application that offers different levels of visibility into malicious activity. This means that without a tool in each layer there will be a huge gap in visibility of how adversaries are interacting with a web application. Although there are advantages and disadvantages to tools at the different layers, each are key to properly securing a web application.

Unfortunately, most security tools that serve to offer insight into an application's behavior have been developed within the Host and Network layers, with very little development in the Application layer. [33] Most web application companies have very little insight into how their users are interacting with their product at the application layer because there are few tools offered that can do that. On the flip side of that almost every web application will have a firewall and AV setup to monitor and protect their network and host layers, respectively.

I provide an overview of a few technologies used in the Network and Host layers to highlight the strengths and weaknesses of each and to provide a context with which to contrast against Application layer technologies.

2.3.1 Network Layer

2.3.1.1 Firewalls

Firewalls serve to protect web applications at the network level by filtering traffic with network features marked as malicious. Common indicators include: traffic being sent to unexpected ports, domains, or IP addresses; unexpected protocols; IP address blacklists; and limiting specific types of output traffic. For example, a simple web server should only receive requests for web pages and should only serve information over the HTTP protocol. A firewall may be installed to ensure that adversaries do not attempt to exploit a server by connecting over SSH or that the server can not be compelled to connect with a client over any protocol other than HTTP.

Firewalls can be placed at many points in a network topology, but are most commonly placed at the edge of an application's network, providing clients with a single point of ingress to the application's functionality. Firewalls can also be placed within an application's topology to manage and monitor traffic between internal nodes.

Firewalls have existed since the late 1980s, with the first paper written about filtering network traffic, screend, for access control by Jeffrey Mogul in 1989. [25] Firewalls can be implemented at almost all levels in the network stack with varying benefits and detriments. The higher up the stack the firewall is implemented the more control and confidence it can have over its rules, however it will also need more specific information and tuning to be effective. The lower down the stack the firewall is located the more flexible it can be, but the less insight it provides and more broad its limitations.

Firewalls are only able to leverage the information contained in network metadata to make operational decisions. This limited set of data can be leveraged to create powerful, yet simple security tools that are able to make low level decisions and block a broad swath of malicious traffic. A firewall is a necessity for any enterprise web application. It has a specific job and does it well. However, firewalls can be misrepresented as a fix-all for web security, when in reality it represents only one way of observing the behavior of an adversary. Firewalls do not analyze the data layer of network traffic and do not monitor the machines that web servers run on.

2.3.1.2 Web Application Firewall

A web application firewall (WAF) builds upon a traditional firewall by performing a very simple analysis on the data portion of a network packet. This means WAF's are working with data from the application layer, but doing so from the vantage point of the network layer. This provides them an extended level of visibility over traditional firewalls, but does not provide them the full visibility of the application layer. Without understanding the context in which the data is being used in the application, WAF's are limited in their ability to discover vulnerabilities. For example, a search bar form field will want to sanitize and/or blacklist certain characters related to SQL injection,

whereas a comment field for a forum will want to sanitize and/or blacklist characters related to Javascript or HTML. A WAF will not typically have the correct insight to understand the difference between the two.

WAF properties described in [31] suggested the idea of "application scrubbing" where firewalls could filter streams for malicious strings at the user-level. Also described by [41] as a "fingerprint scrubber". These early definitions paint an accurate picture of the best uses of WAF's, even in modern technologies. They are most powerful when utilized with simple parameters. Common tasks including searching HTTP traffic for malicious input commonly used to perform XSS or SQLi. WAF's can block or filter out blatant attempts at XSS exploitation and can be tuned to be more aggressive or relaxed in specific instance.

While a WAF is more sophisticated than a traditional firewall, it is still limited in its visibility. Much like traditional firewall though, WAF's are often viewed as able to perform more tasks than they are designed to. Just because a WAF can use a blacklist to identify and block XSS strings, that does not mean that the corresponding web application is impenetrable from XSS attacks. A WAF is another useful tool in a web application's arsenal, but should be implemented as a piece of a defence-in-depth strategy and not as a crutch.

2.3.2 Host Layer

2.3.2.1 Anti-Virus

Anti-virus (AV) tools are used to scan the a host system at either a user or systems level for malware. Traditionally, AV systems used *signature based*, that is they would compare the binaries on the host to a list of known bad and good binaries, to determine whether a binary was malicious or not. With the advent of *packers*, tools designed to bypass signature based scanning by compressing executables so that

they have a different signature, AV tools have been forced to switch to *behavioral based* detection of malware, which analyzes how a binary executes in a sandboxed environment. [30] There are many different AV solutions provided to both consumer and enterprise markets by companies such as Symantec, Kaspersky, Comodo, Avira, ESET, Microsoft, and many more.

AV systems have come under heavy fire in the last couple years by research conducted by Google Zero's Tavis Ormandy. Ormandy has analyzed many AV solutions and found that in many cases the software is so buggy that it can actually make a user *less* secure. [35] [36] [37]

All that aside, the visibility of antivirus is restricted to the metadata of processes and executables running on the host. In the context of web application security, this means that AV is really only effective and detecting malware being loaded onto or running on the same system as a web application. It has no visibility into how the running web server is being used, or abused. A legitimate application could be used in malicious ways, and this is out of the scope of AV software. Although a useful tool that provides a floor of host security when implemented correctly, antivirus does not provide visibility into the logic of a web application.

2.4 Application Layer

The application layer differs most from the host and network layers in that its perspective come from within the application that instead of trying to peer in at the application or searching for symptoms of attack that have leaked out of the application. This perspective has both its advantages and disadvantages when trying to monitor malicious activity.

On the plus side, by virtue of its definition it has direct access to the source code of the application, allowing tools residing in the application layer to have a complete

understanding of the context of the behaviors being monitored. This access can be used to observe the context of user activities, and also engage the application to perform functions in direct response to malicious behavior.

For example, where a network firewall may observe traffic from a blacklisted IP address and then choose to block that traffic, an application layer tool could observe a blacklisted string being attempted to be submitted to a search field and then log the user out.

Access to the source to understand the context of user activities can be a very powerful asset, but the downside of this is that it requires integration with source code, maintenance, and setup. The cost of these integrations may differ from application to application, but no matter where an application lies on this spectrum, the effort to implement and maintain extra source code is going to be greater than installing AV on hosts or including a network monitoring box in a network topology and monitoring the output.

While there are many security technologies that reside in the application layer, none of them provide insight into user activity or report security events. Technologies like encryption, authentication, and session management represent a slice of what is available at the application layer to improve the security of web application, but none of these tools help administrators understand the malicious activity of adversaries.

2.4.1 Application Level IDS

App level IDS differs from traditional solutions by operating in the application layer - where user data is processed - as opposed to over the network or on the host. Instead of trying to determine if incoming data over the network is malicious, or waiting to find something malicious on the host, app level IDS can monitor how users are operating the application to determine if they are doing so maliciously.

App level IDS focuses on user behaviors within the application by monitoring various points in the code of an application that user will interact with and observing any unexpected behavior at these points. Although all unexpected behavior does not lead to malicious activity, it can be analyzed for patterns of malicious activity.

App level IDS is more effective at determining intent of users and abuses of applications than traditional defenses. It does however, require significant more work to install, maintain, and instrument to be used effectively. This trade off in work and time may only be worth it if it provides a significant improvement in performance - specifically improved accuracy through false-positive and false-negative rates.

2.5 AppSensor

AppSensor is an application level IDS developed by Open Web Application Security Professionals (OWASP). OWASP is a non-profit organization focused on delivering open-source solutions to improve and support the web application security community. Most known for the OWASP Top 10, a list of the top 10 most important web vulnerabilities, OWASP also supports projects such as the Zed Attack Proxy (ZAP) and the ModSecurity Ruleset in addition to organizing local chapters, conferences, and online resources.

AppSensor is a service that given an input of information from sensors embedded in an application, will store and process them, and then when it has determined an attack has occurred will return notifications and responses. Although it is written in Java, it can be incorporated with any web application via a number of different integration methods.

Chapter 3

RELATED WORK

3.1 RASP

Runtime application security protection (RASP) is an emerging brand of solutions for web application security that aims to detect attacks at runtime and provide feedback to the web application so that it can properly handle any malicious activity. RASP tools often run at the web server layer, integrating themselves within the Java JVM or the .NET execution environment. By monitoring the function calls throughout the running web application, RASP tools can detect attacks within the proper context of the web application, *without* requiring deep integration within the source of the application. Although not truly running in the application layer, integrating with the web server's execution environment is as close to the application layer a tool residing in the host can get.

RASP technology is still very young, and the solutions in this space are still quite varied. Gartner [22], Contrast [4], HP Fortify [1], Waratek [5], Arxan [19], and Prevoty [2] are all different vendors competing in this space. Proponents of RASP point to the out-of-the-box simplicity of integration and the level of control and power by running in the web server. Those who are less enthused about the technology claim that RASP tools often provide the same protection as a WAF, but in a more complex and expensive manner.

While community opinion about RASP is still developing, the fact of the matter is that RASP tools do not fix vulnerabilities, as many claim they do, but rather add another layer of security in front of them. In some cases this security may be closer to the WAF claims, where a RASP tool may lie in front of a database and simply run

regex against the commands being submitted to identify SQLi. On the other side of the spectrum, some tools execute payloads being submitted to various parts of the application in a virtual environment and then analyze the result of the payload all in real time to determine if a payload is malicious. In either case the vulnerability remains behind the RASP protection. Regardless, the RASP methodology can provide an effective tool when used in a defense-in-depth manner. Additionally, RASP tools can be leveraged to provide highly localized monitoring to be analyzed by another system. [20]

3.2 Salesforce

Salesforce has created an event monitoring system built around their Salesforce applications. [23] Salesforce apps are able to generate security events, which can be monitored using the Salesforce Shield Event Monitoring app. Salesforce supports events such as logins, data access, uploads, downloads, geoip information, and over 50 other types. The Event Monitoring app allows an administrator to monitor and query past results, but also enables them to set up a Transaction Security Policy. [6] A Transaction Security Policy enables administrators to perform actions such as block an action, logout, require two factor authentication, or notify when any user creates a specified number of security events. For example, a Transaction Security Policy could be set to block any user from downloading information from a specific database or could alert an administrator if there were more than than 4 (or however many) failed logins from a single user.

The Salesforce Shield Event Monitoring app and Transactional Security Policy provide a similar application level monitoring and control system to that of AppSensor, but built into the Salesforce application environment.

3.3 Repsheet

Repsheet is an open source project similar to AppSensor that processes streams of security events to decide whether to black, white, or grey list users. [3] The driving goal of Repsheet is to identify bots and add layers of resistance to the workflow of identified possible bots to force the user to prove they are humans. While Repsheet has a very similar design to AppSensor, the bot centric goals of Repsheet often produce a much more specific subset of rules than AppSensor might. Additionally, Repsheet limits its responses to adding users to lists, whereas AppSensor leaves responses as open ended options for the administrator of the system. These systems could be used in place of each other, or in parallel with each other. Sensors from AppSensor could actually feed the input of Repsheet to help it make decisions.

Chapter 4

REQUIREMENTS

4.1 AppSensor Architecture Overview

The AppSensor System is made up of several components that can be split into four different groups for simplicity: sensors, event manager, data stores, and analysis engines. Figure 4.1 shows that there are one to many sensors, a single event manager, three data stores, and three analysis engines. Each component is modularized and can be configured with several different options.

4.1.1 Sensors

Sensors are the event generating pieces of code that reside in the actual application. These are placed at points of interesting user behavior that is worth monitoring. For example, a sensor may be placed at a point in the application that checks that submitted monetary values are positive and, if not, rejects the request. Here, it would be interesting for the sensor to report back that user X tried to submit a negative amount in a monetary field.

To do this only two pieces of information are necessary: an ID representing the user, whether that be a username, email, or other unique user ID being used, and a label representing what the event was. In this case I may use the label "IE1", where IE stands for Input Exception, and IE1 represents "Negative Monetary Amount Input". The event label is just a string and can be anything, but there is a recommended list of label codes to use from the AppSensor project. [15] This information is then relayed from the application code to the AppSensor server using one of several different possible communication methods, such as a REST API request.

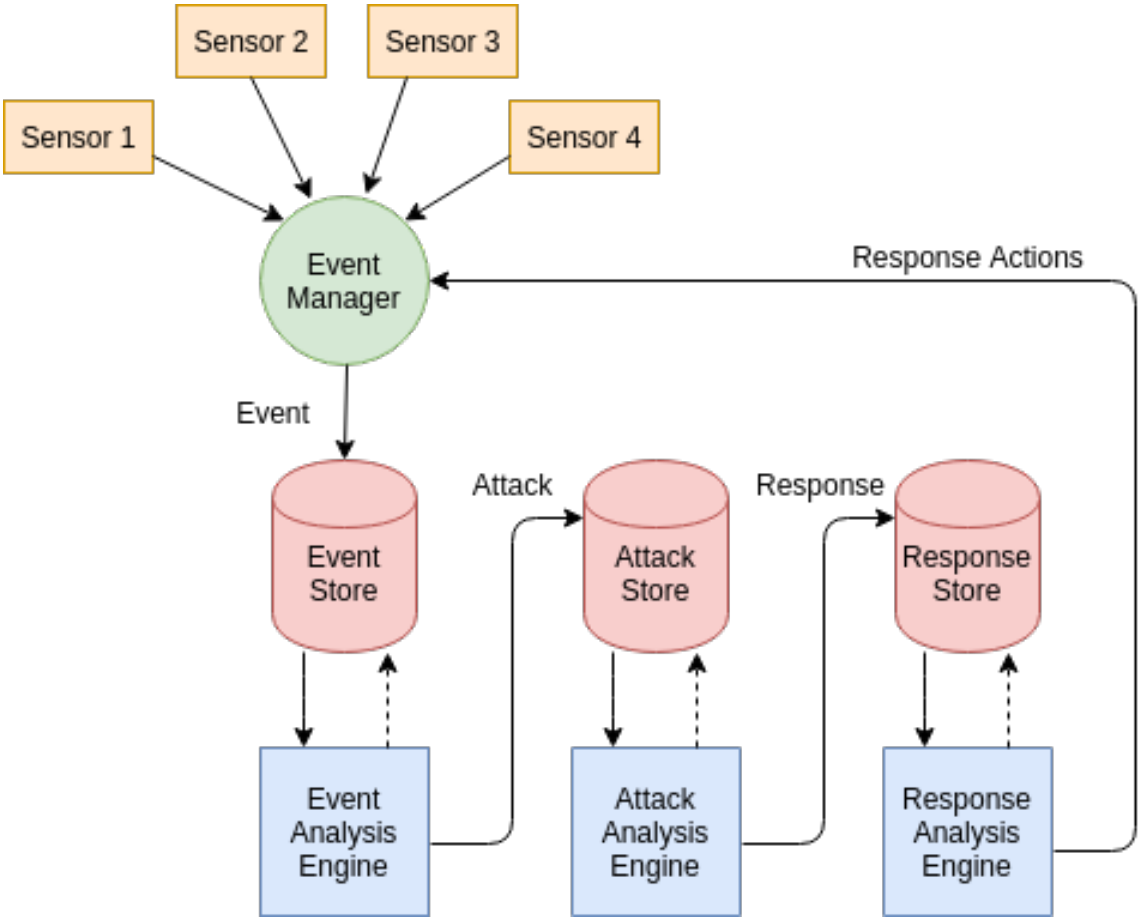


Figure 4.1: Overview of AppSensor architecture.

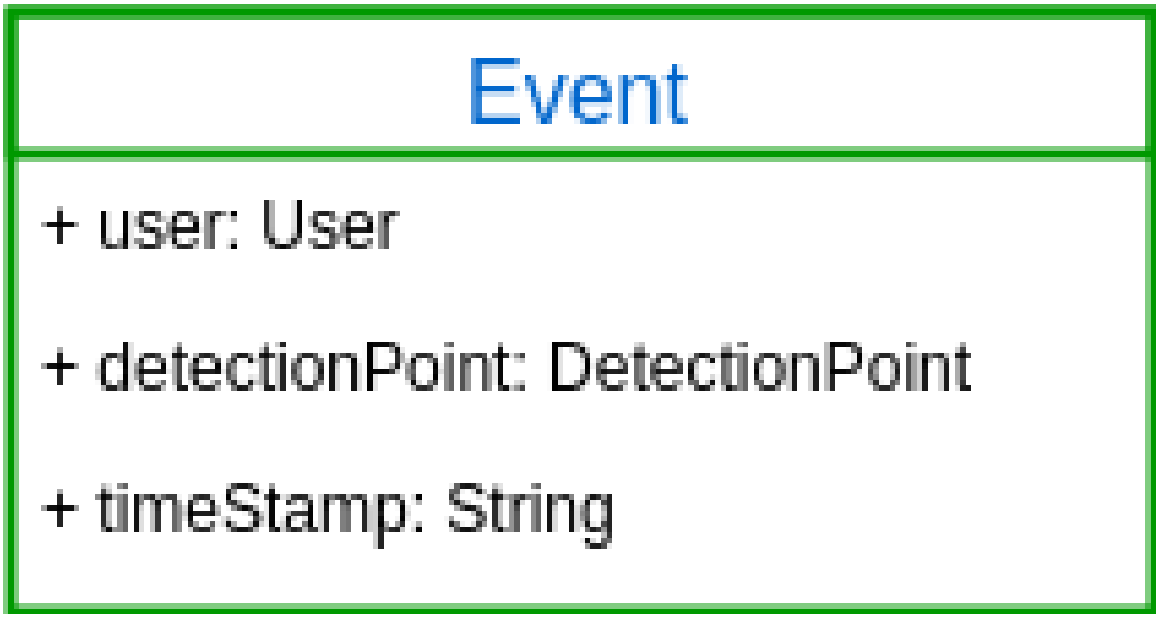


Figure 4.2: UML diagram of Event class showing relevant member variables.

4.1.2 Event Manager

There are several components that make up the *event manager*, but for simplicity they can all be considered to be one entity. The event manager runs on the AppSensor server and receives messages from the sensors, generates events, and then adds them to the event store. The event manager can be configured to communicate with the sensors using several different methods including: REST API, SOAP API, Thrift, Kafka, ActiveMQ, RabbitMQ, as well as with Java on a local instance.

4.1.3 Data Stores

The *data stores* hold three different data structures that are accessed by AppSensor: events, attacks, and responses.

Events hold the information generated from sensors in the application source code and are stored in the Event Store. Events contain information including a detection point which stores the label, what user caused it, and the time that it occurred.

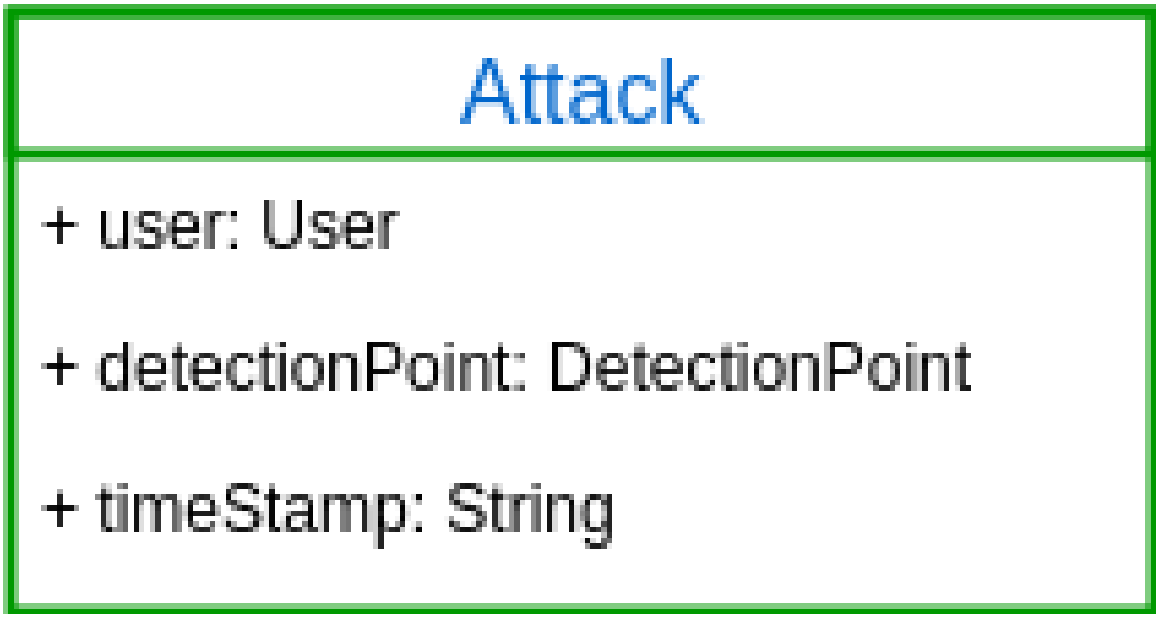


Figure 4.3: UML diagram of Attack class showing relevant member variables.

Attacks are generated from the Event Analysis Engine and are stored in the Attack store. Attacks contain information about the user that caused it, what Detection Point triggered it, which sensor it occurred in, and what time it occurred.

Responses are created by the Attack Analysis Engine and are stored in the Response Store. Responses contain information about what user it is in response to and the actions that need to be carried out in response to the attack.

Data stores can be configured as one of several different modules. Currently modules include support for in-memory storage, file-based storage, InfluxDB, JPA2, ElasticSearch, Mongo, and RIAK.

4.1.4 Analysis Engines

There are three different *analysis engines* that process information from the data stores: event analysis engine, attack analysis engine, and response analysis engine.

The event analysis engine registers as a listener to the event data store, so that

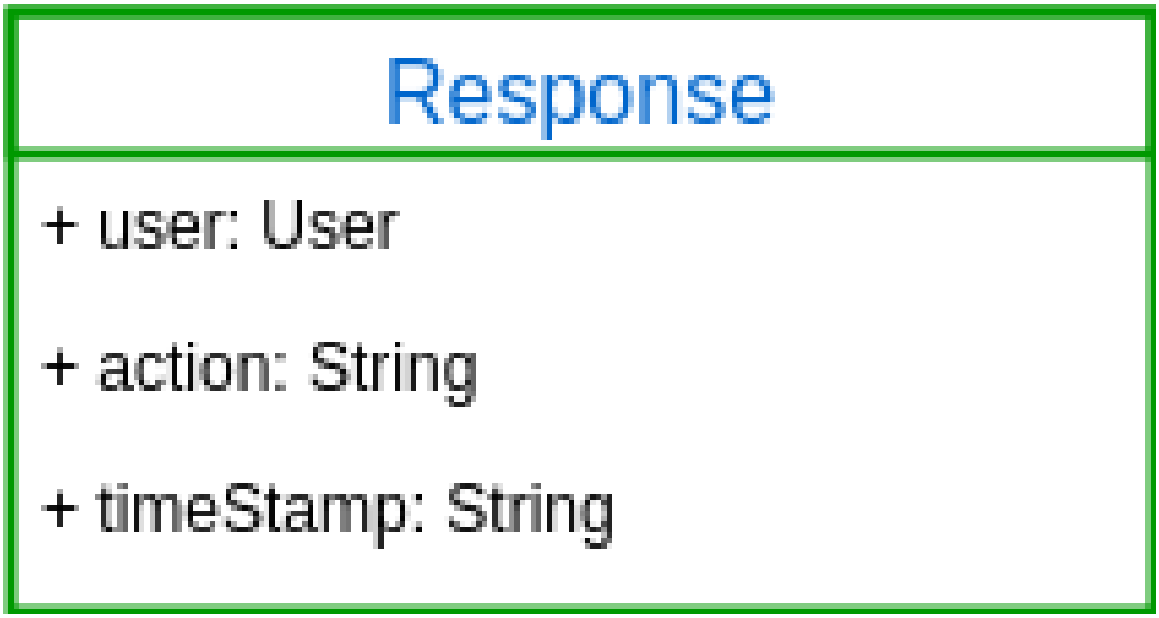


Figure 4.4: UML diagram of Response class showing relevant member variables.

each time a new event is added to the data store the event analysis engine can process it. After processing each newly added event, the event analysis engine will generate an attack if it has determined the event has caused an attack and add that attack to the attack data store.

The attack analysis engine registers as a listener to the attack data store, so that each time a new attack is added to the data store the attack analysis engine can process it. The attack analysis engine will determine what responses need to be created from the newly added attack, and then add those responses to the response data store.

The response analysis engine registers as a listener to the response data store, so that each time a new response is added to the data store the response analysis engine can process it. The response analysis engine will process each new response added to the data store and determine what actions need to be performed and then either handle those actions itself or delegate the actions to another process.

Each analysis engine has a similar structure. The AttackAnalysisEngine class implements the AttackListener interface, the ResponseAnalysisEngine class implements the ResponseListener interface, and the EventAnalysisEngine class implements the EventListener interface. The listener interfaces each define only a single method onAdd, which accepts a single parameter of matching type. For example, the AttackListener defines a method onAdd(Attack attack).

The designed purpose of the attack analysis engine is to add the appropriate responses to the response data store, and the response analysis engine processes the response actions. In the case of the rules-based engine that I will be building the attack analysis engine will only add responses to the attack data store and the response analysis engine will simply log the responses when they are added to the response data store. Additional functionality can be added by extending the set of analysis engines. The most interesting analysis engine is the event analysis engine, which must determine if an attack has occurred by processing the incoming stream of events. When referring to the rules-based engine in the future, I will mean specifically the event analysis engine of the rules-based analysis engines.

4.2 Event Analysis Engine Structure

The event analysis engine's purpose is to determine if an attack has occurred by processing the incoming stream of events from the sensors as they are added to the event store. The heavy lifting algorithm that does this processing is found in the *analyze* method of the event analysis engine. When an event is added to the event data store and the onAdd(Event event) method is called, it immediately calls the analyze(Event event) method. The analyze method will process the event and either generate an attack or not.

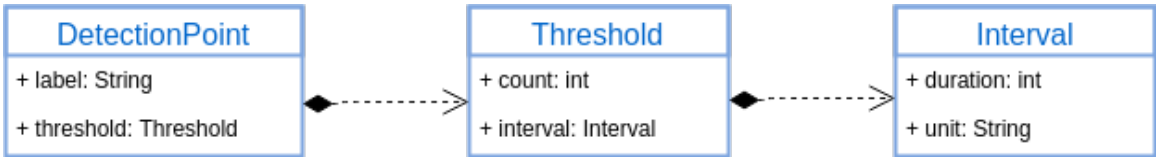


Figure 4.5: UML diagram of DetectionPoint, Threshold, and Interval classes showing relevant member variables.

4.2.1 Reference Event Analysis Engine

Understanding how the reference event analysis engine works will shine a light on the existing structures and patterns as well as to create an outline for how the rules-based event analysis engine.

The reference event analysis engine's analyze method works by counting the number of events from a specific sensor, and then generating an attack if there are more events than a specific threshold. The reference engine does this by using a preconfigured list of *detection points* that are described in a configuration file. Detection points are a combination of a sensor label ("IE1" for example) and a corresponding *threshold*. A threshold object represents X number of events occurring in Y time, where the number of events is stored as a member variable along with an *interval* object which holds an amount of time. For each sensor in the target application, there is a corresponding detection point that is defined. For example, if a sensor is creating events with the label "IE1, then there will be a detection point defined with label "IE1" and a configured threshold amount.

Each time the analyze method is called it does three things. It first gathers all related events by querying for existing events in the event store with a matching detection point label. Second, it filters out events that do not fall within the interval of the detection point's threshold. Finally, with a collection of events that were created by the same sensor and occurred within the correct interval of time, the analyze method simply counts the size of the list and compares it against the threshold

amount. If there are more events than the threshold defines, then the an attack is created.

For example, assume there is a detection point for label "AE1" with a threshold of 5 events in 10 minutes". If the analyze method is called with an event with a label of "AE1", it will first query the event analysis engine for all events that have been previously logged with a matching label of "AE1". To compare the number of events with the detection point's threshold, however, only the events that occurred 10 minutes before the last event need to be counted. Additionally, if an attack from this detection point already occurred recently, the events that were counted for that attack should not be counted again. So the analyze method filters out all events that occurred before 10 minutes prior to the last event as well as any events that occurred before the last attack. Finally, with a list of events that occurred within the proper time constraints, if there are more than 5 events than an attack is created and added to the attack store.

While the reference engine's threshold model is relatively simple, the same basic outline for the analyze algorithm can be used in the rule-based engine:

First, *gather related events*. Before any analysis can be done the algorithm needs all of the proper information.

Second, *process events*. This may seem like a vague step, but the idea is that work needs to be done on the events so that they are in a proper state to be evaluated. In the case of the reference engine this meant filtering out events that were not in the proper interval of time.

Finally, *check attack condition*. The final step is to take the result of the processed events and compare it with the attack condition to see if an attack should be generated. In the case of the reference engine this means counting the filtered events and comparing them to the threshold.

Additionally, the existing detection point and threshold structures will need to be utilized when building more complex combinations for rules.

Chapter 5

DESIGN

The model for the rule-based event analysis engine is based off the reference event analysis engine. Using the reference engine as a template, the basic outline of the rules-based event analysis engines analyze algorithm will be structured like the pseudo-code below.

```
for each rule:
        collectRelatedEvents()
        processEvents()
        if attackConditionsMet:
                createAttack()
```

First, the algorithm will collect all events that are related to the evaluation of the rule. Second, the events are processed to determine if the rule was triggered. Third, if the rule was triggered an attack is created.

This chapter will overview the design behind each of the components that built up the analyze method. Before the components can be discussed however, the structure of the rule must first be defined.

5.1 Rule Structure

A *rule* has several pieces, with the most basic unit being a *monitor point*. A monitor point is essentially the same thing as a detection point, except that it cannot trigger attacks on its own. Where a configured detection point stands alone and will generate an attack when its threshold is crossed, a configured monitor point can only be a part of a rule and does not generate an attack when it's threshold is crossed.

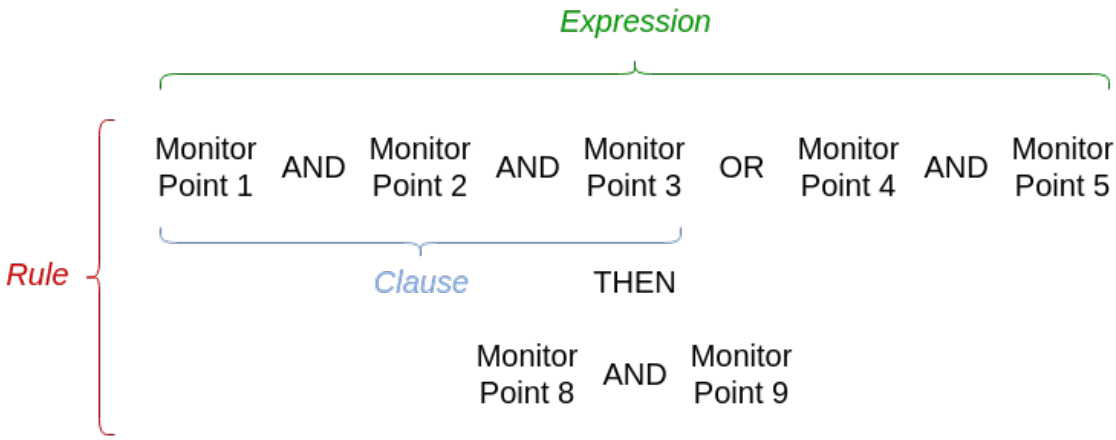


Figure 5.1: Representation of rule structure.

Rather only when the proper configuration of monitor points in a rules logic are triggered will the rule then generate an attack.

A rule is made up of one or more *expressions*, where an expression is a group of monitor points along with the boolean AND and OR operators. Expressions are separated within a rule in chronological order by the THEN operators. A rule will generate an attack only if *all* of its expressions evaluate to true. All expressions must be triggered within an interval of time called the rule's *window*. The chronological nature of expressions means that an expression cannot be triggered until the expression before it has been triggered. An expression can only begin after the end of the previous expression, that is expressions cannot overlap.

The rules window must be greater than or equal to the sum of the windows of each of its expressions. When the rules window is greater than the sum of its expressions, it provides flexibility on when the expressions must actually be triggered. This wiggle room means that instead of just one possible interval of time for the expressions to be true, there are many possible combinations that could lead to each of the expressions evaluating to true.

This can be seen in Figure 5.2 where the size of the rules window is greater than

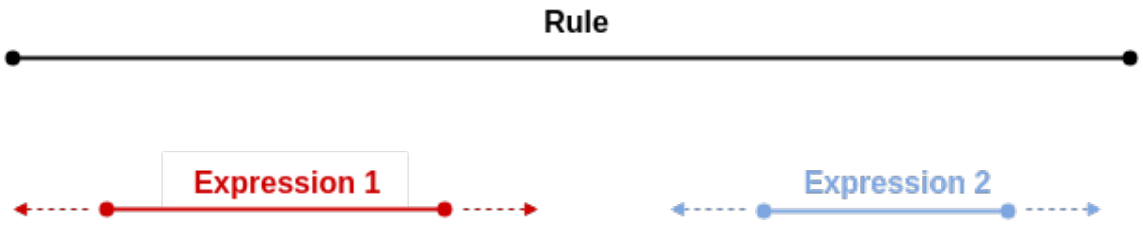


Figure 5.2: The windows of two expressions beneath the window of a rule.

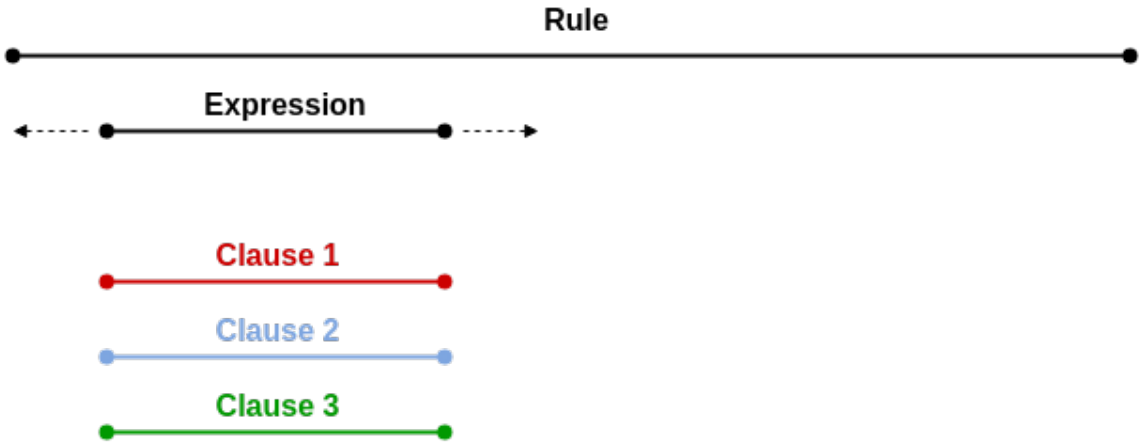


Figure 5.3: The windows of three clauses beneath the window of an expression.

the sum of the windows of Expression 1 and Expression 2. This extra time means that there are many possible intervals that result in Expression 1 occurring before Expression 2.

An expression is made up of one or more *clauses*, where a clause is a group of monitor points along with AND operators. Clauses are separated within an expression by the OR operator. An expression will evaluate to true and be triggered if *at least one* of its clauses evaluates to true. A clause must be triggered within an interval of time called the expression's window. A clause shares the window of its parent expression.

As can be seen in Figure 5.3, the size of the expressions window determines the window of its clauses.

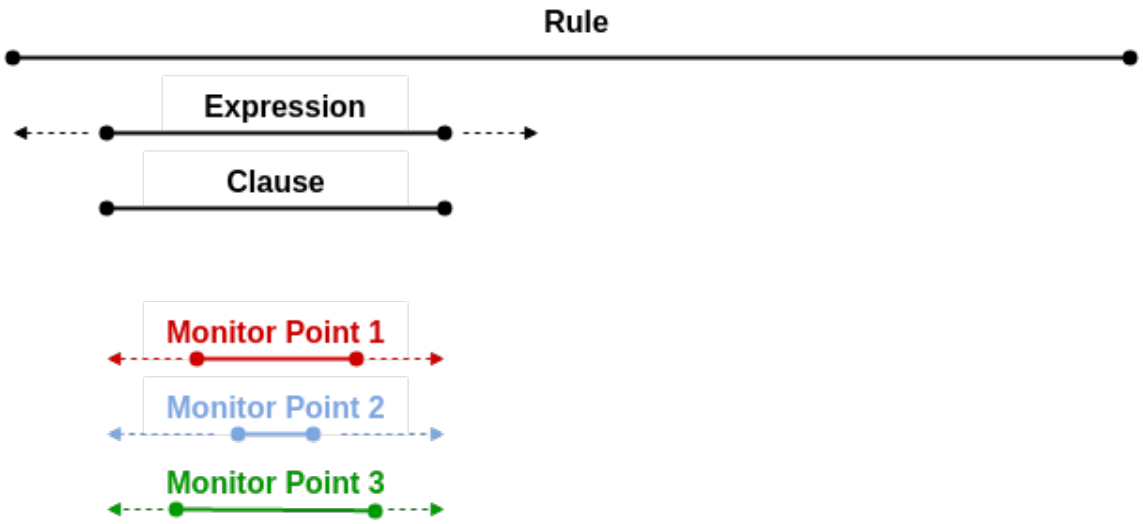


Figure 5.4: The windows of three monitor points beneath the window of a clause.

A clause is made up of one or more monitor points, where a monitor point represents a specific sensor. Monitor points are separated within a clause by the AND operator. A clause will evaluate to true and be triggered only if *all* of its monitor points are triggered. Each monitor point has a window defined by its threshold. This window must be smaller than or equal to the window of its parents clause. Because a monitor point can have a smaller window that its parent clause, there are many possible times at which a monitor points window could be evaluated at.

This can be seen in Figure 5.4 where the three child monitor points each have a unique window that is able to slide up and down the window of their parent clause.

The operators of the rule are implied in the design of the objects structure. A rule object is built up of a list of expressions. The THEN operator is implied between each expression in the list. An expression object contains a list of clauses. The OR operator is implied between each clause in the list. The clause object contains a list of monitor points. Between each monitor point in the list is an implied AND operator. Thus the structure of the rule holds the logic as well.

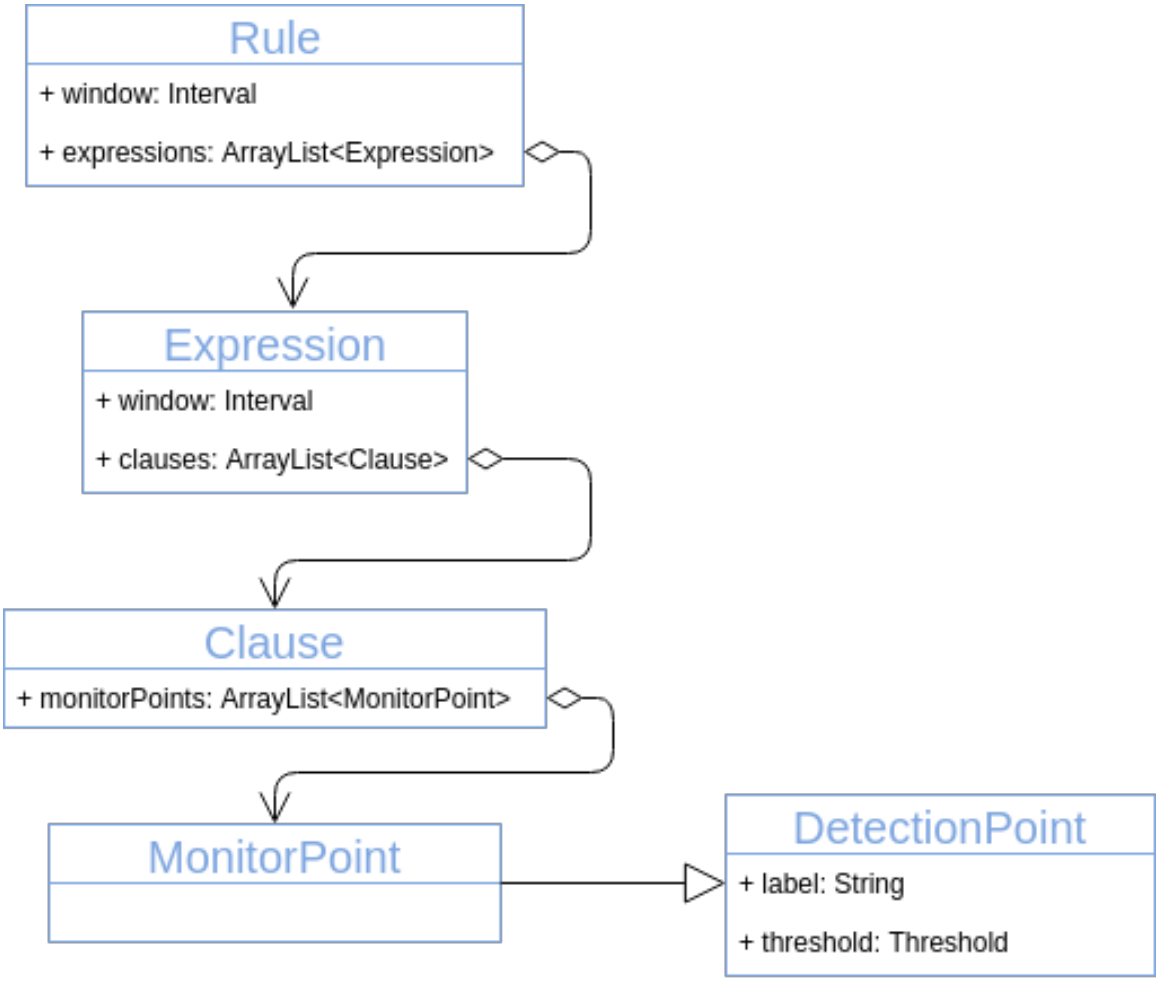


Figure 5.5: UML diagram of relevant fields within the classes of the rule structure.

5.2 Algorithm Outline

5.2.1 Collect Related Events

This first stage of the algorithm is to collect all events needed to properly analyze the rule at hand. Because a rule can be made up of multiple monitor points, events of more than one sensor label will need to be collected. Events that match any of the monitor point labels will need to be queried for. To simplify the process, the query also includes a time that all events will have to be after to be included. This time is either the last attack time or the rules window amount of time before the last event, whichever is later.

5.2.2 Process Events

Now that all of the proper events have been collected, the algorithm needs to determine whether that set of events matches the pattern of the rule that triggers an attack.

Where the reference engine had a strictly defined interval of time that needed to be checked, the rules-based engine has windows of time with many possible intervals that need to be checked. The loose nature of the rules structure means that there are many possible combinations of events that could match the triggering pattern of an attack, and the algorithm may need to check all of them.

The first step to analyzing the rule is to determine whether each of the monitor points evaluates to true or not. Recall, a monitor point has the same structure as a detection point, so to evaluate a monitor point the algorithm just needs to compare the number of events in the monitor points interval against the count of the threshold - just like the reference engine does. The problem is that we dont know the exact interval of time to use to evaluate the monitor point, it could be any interval of time

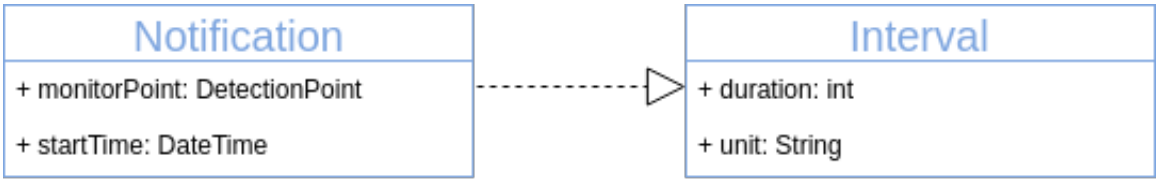


Figure 5.6: UML diagram of Notification class and relevant fields.

within the window.

5.2.2.1 Creating Notifications

To address this problem we will check *all* intervals. Instead of checking whether a monitor point was triggered at a single interval of time, the algorithm will check *all* possible intervals that a monitor point could have been triggered. If the monitor point was triggered, that is if in there were more events than the monitor points threshold count within the interval of time being checked, then we create a new object called a *notification* and add it to a list. A notification represents a triggered monitor point and holds the interval of time over which it was triggered. It can be compared to an attack object, in that when a detection point is triggered it creates an attack, when a monitor point is triggered it creates a notification.

An example of creating notifications can be seen in Figure 5.7. In the example the monitor point is labeled as "mp1" and its interval is represented by the red bar. The timeline beneath the monitor points interval shows the distribution of events over time. The interval slides down the timeline, checking groups of events that fall within its time bounds. In this example, the monitor points threshold count is three. So each time the interval slides over three events, a notification is created. By the end of processing events for the monitor point, four notifications have been created.

Doing this for each of the monitor points will give us a list of notifications for different types of monitor points. These notifications will help us to evaluate the base

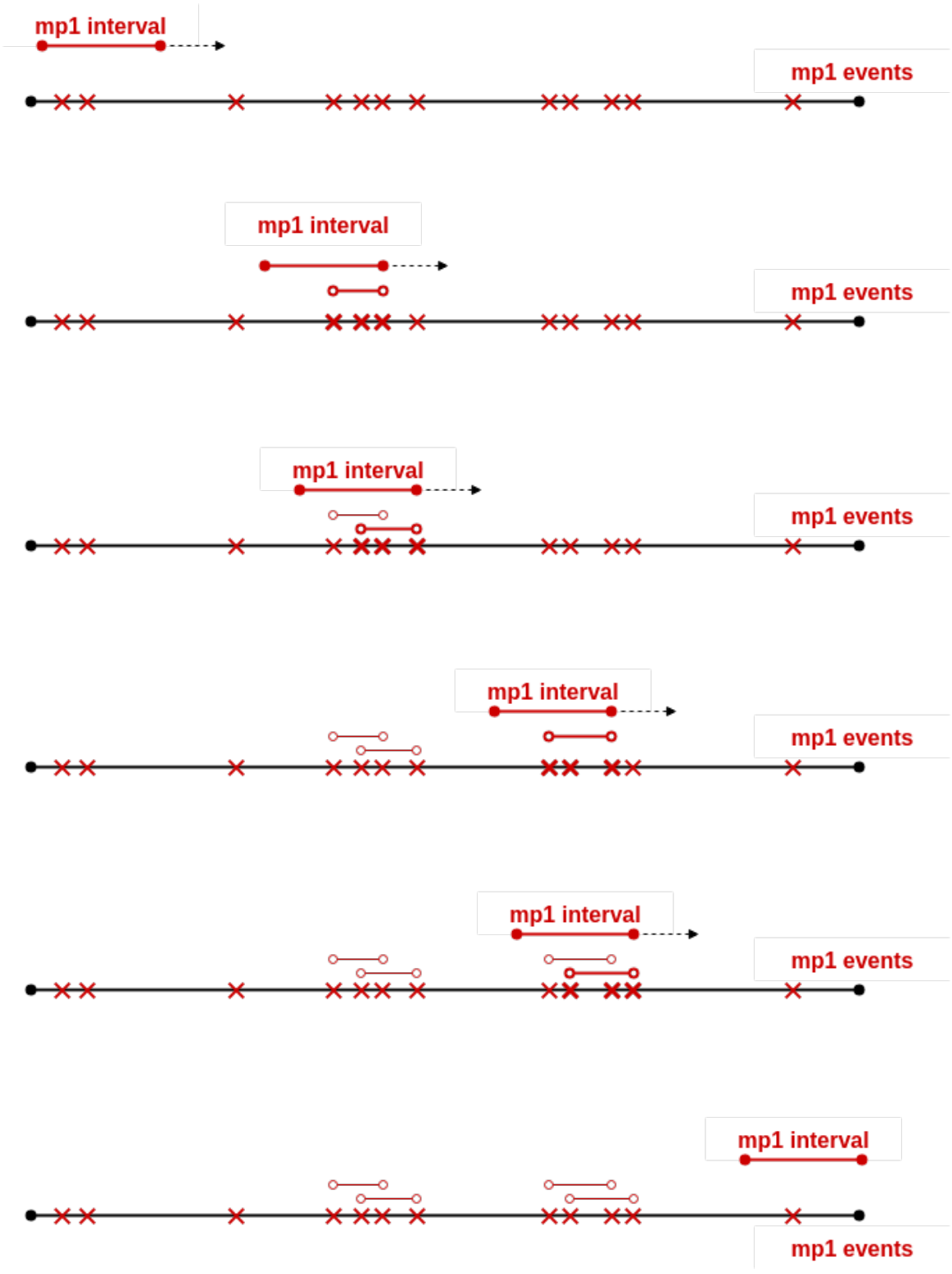


Figure 5.7: Example of creating notifications.

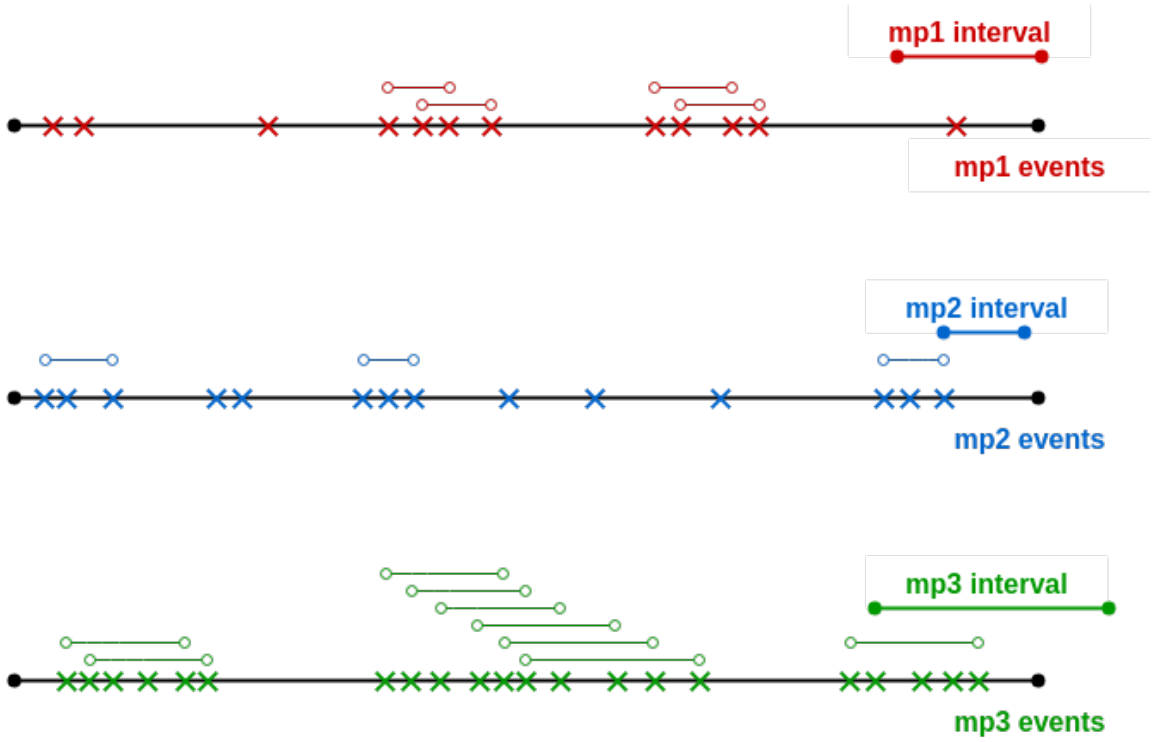


Figure 5.8: Example of notifications for three different monitor points.

unit of the rule, the monitor point, which we can use to begin to evaluate the entire rule. Figure 5.8 shows notifications created for each of the monitor points.

With notifications created for each monitor point in the rule, the rule can now be evaluated by starting the "sliding window" of the first expression at the first notification. In the example shown in Figure 5.9, the expression being evaluated has only one clause, "MonitorPoint1 and MonitorPoint3". MonitorPoint1 is represented by the red notification bars and MonitorPoint3 is represented by the green bars. As shown in the example, the first expression is going to slide down the timeline until it evaluates to true.

For the first expression to evaluate to true, at least one of its clauses must evaluate to true. For a clause to evaluate to true, all of its monitor points must have been triggered. That is there must be a matching notification for each monitor point in the clause. Because a clauses window is the same window as the expression, to evaluate a

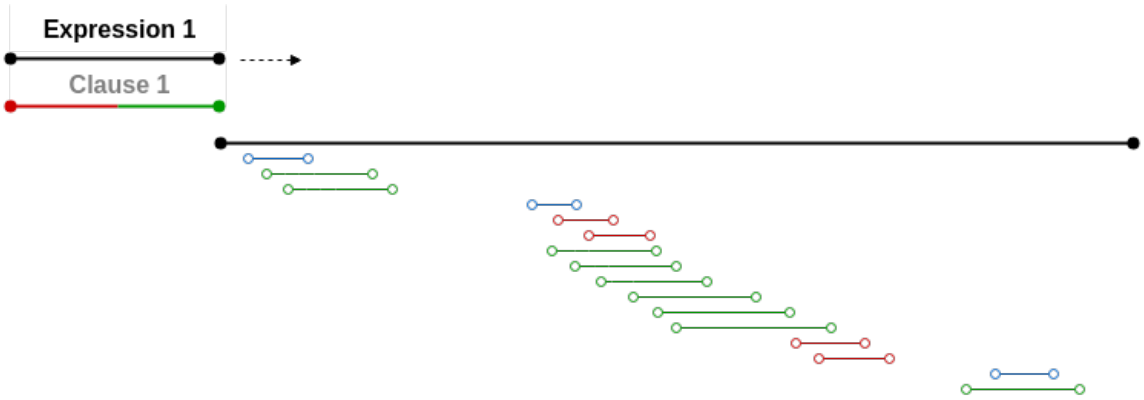


Figure 5.9: Example of using the sliding window to start evaluating a rule.

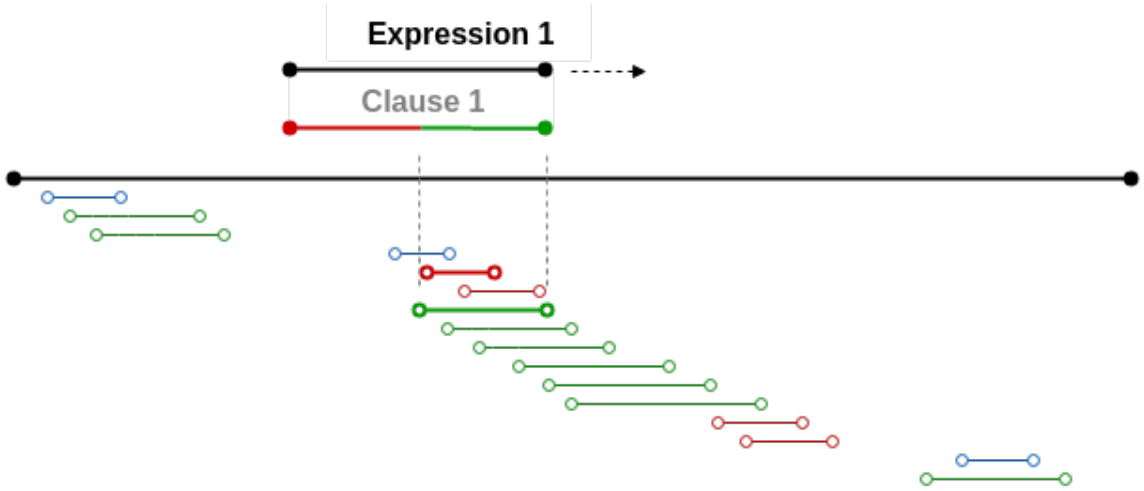


Figure 5.10: The expressions window slides above a set of notifications that cause the clause to evaluate to true.

clause the algorithm just needs to check whether all of the clauses monitor points have matching notifications within the window of the expression. If there are notifications from each monitor point within that window, then the clause evaluates to true, and thus the expression evaluates to true. If not, the next clause is checked. If each clause evaluates to false, then the expression, and thus the rule, also evaluate to false.

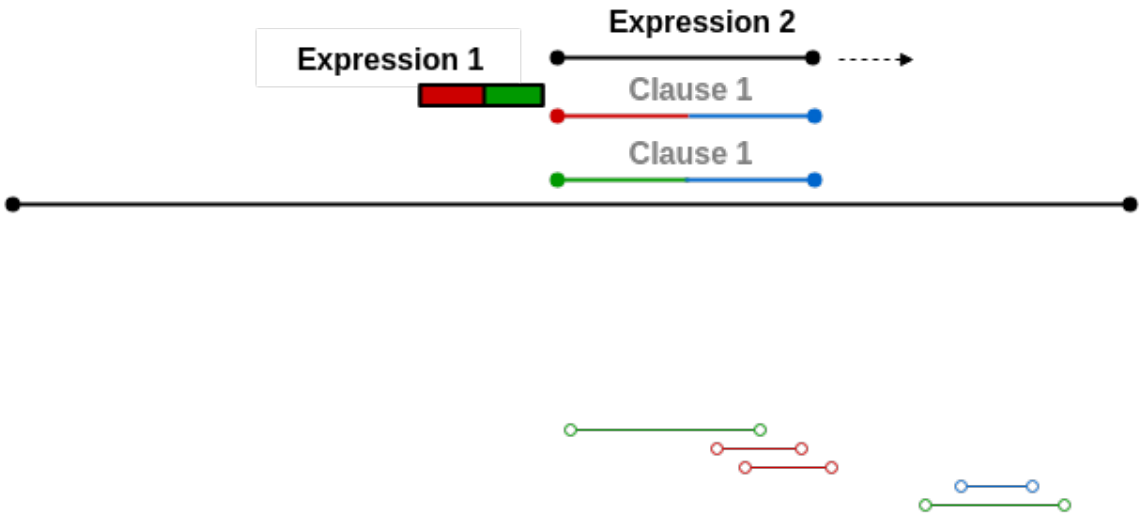


Figure 5.11: With the first expression evaluating to true, the second expression starts to slide where the first ended.

5.3 Evaluating Attack Conditions

When an expression does evaluate to true, it is important to note where exactly within its window the clause was triggered, so that there is a starting point for the next expression. In this case, we can treat the list of notifications as a queue, and pop off each notification that occurred before the end of the triggered clause. Now the queue of notifications is ready to run through the process again for the next expression.

In Figure 5.11 the first expression has already evaluated to true, and the second expression begins its evaluation of the remaining notifications. Only notifications that begin after the end of the evaluated expression one can be considered.

If each expression evaluates to true within the window of the rule, then the rule evaluates to true, and an attack is generated.

Chapter 6

IMPLEMENTATION

The rule-based analysis feature is built up of several components. As described in the previous chapter, there are actually three separate engines that comprise an analysis engine: the event analysis engine, the attack analysis engine, and the response analysis engine. In addition to the engines, there will be structures to support the rule design: rule, expression, clause, monitor point, and notification. The most interesting component, however, is the event analysis engine which processes the incoming events and determines if an attack should be created. The algorithm that drives the event analysis engine resides in the analyze method of the analysis engine, and this chapter will focus on that algorithm.

All methods referenced can be seen in Appendix E

6.1 Analyze Algorithm

The analyze method itself is relatively simple. The method first finds all configured rules that relate to the *triggerEvent*, the event most recently added to the event store which triggered the analyze method. It then iterates through each related rule and uses the checkRule method to evaluate them. The checkRule method uses the rule and the *triggerEvent*, to determine if an attack occurred. If the checkRule method returns true, then an attack is created.

A rule is related to the *triggerEvent* if there is a chance the event triggered the rule. This means that the *triggerEvent* would have to match with one of the monitor points in the rule's last expression. Recall that an Event has a DetectionPoint field used to indicate the sensor it was sent from. If the label (i.e. "AE2") of the detection

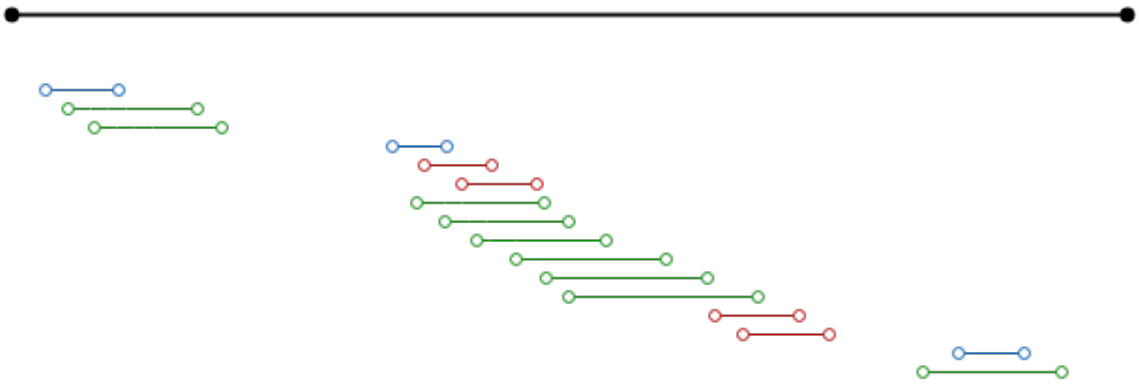


Figure 6.1: Example of how the notifications queue would be sorted.

point matches the label of a monitor point, then it can be said that the event matches with the monitor point. Because the *triggerEvent* would be the last event in analysis if it triggered a rule, it only needs to match with monitor points in the rule's last expression.

6.1.1 Checking Rules

The checkRule method contains the core logic of the algorithm. The first thing this method does is call getNotifications, which takes care of querying for all of the events related to the rule and then building a queue of the notifications, called *notifications*. This method will be detailed later, but for now assume there is a queue of notifications that represent whether any of the monitor points in the rule currently being evaluated were triggered. The *notifications* queue contains notifications from any monitor point in the rule and is sorted by the ending time of the notification.

In Figure 6.1, an example of how the *notifications* queue might look can be seen. The different colored bars represent notifications for different monitor points. The queue is sorted by end time of the notification.

The next step after creating the *notifications* queue is to use the "sliding window" to check all of the possibilities for each expression. To accomplish this, the

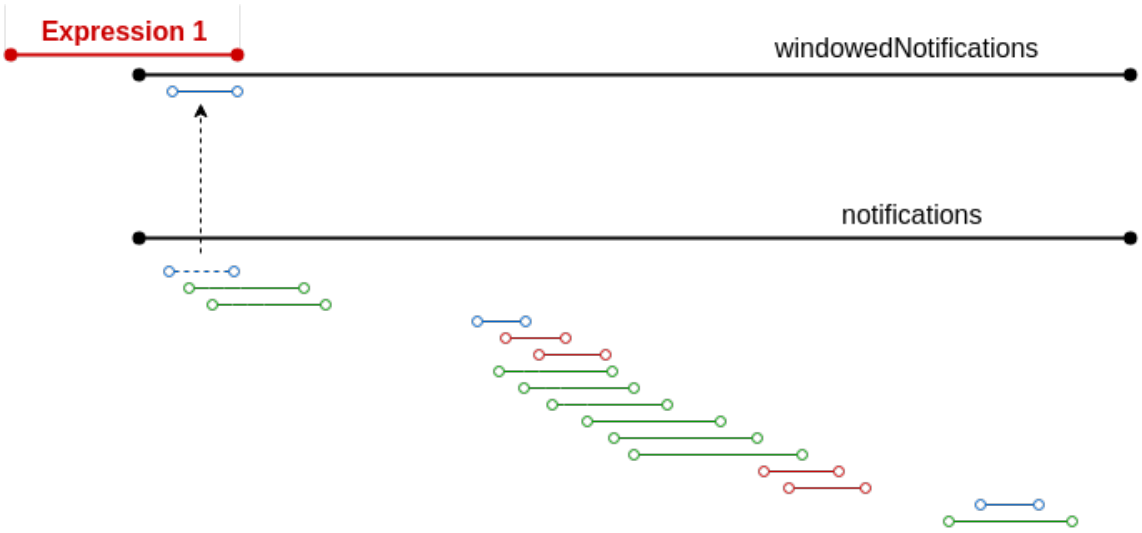
first expression in the rule is set to *currentExpression*. *currentExpression* will then be continually evaluated against a subset of the *notifications* queue, the subset representing the notifications under the sliding window. This subset is structured as another queue, windowedNotifcations.

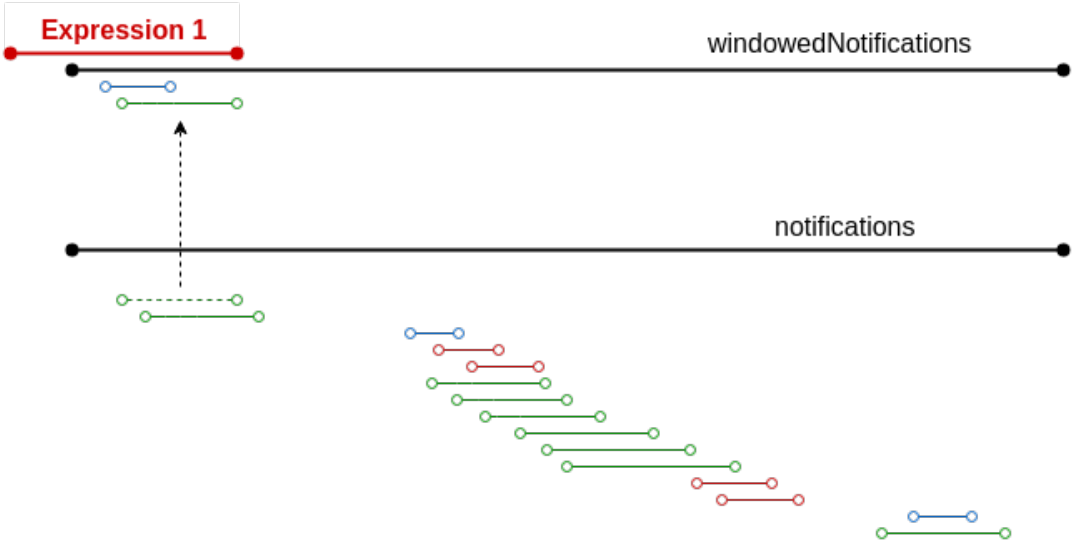
As currentExpresssion slides down the timeline, notifications from *notifications* are popped out one at a time and added to *windowedNotification*. Each time a notification is added to *windowedNotification*, *currentExpression* checks to see if the set of notifications in *windowedNotification* causes it to evaluate to true. This is done with the checkExpression method.

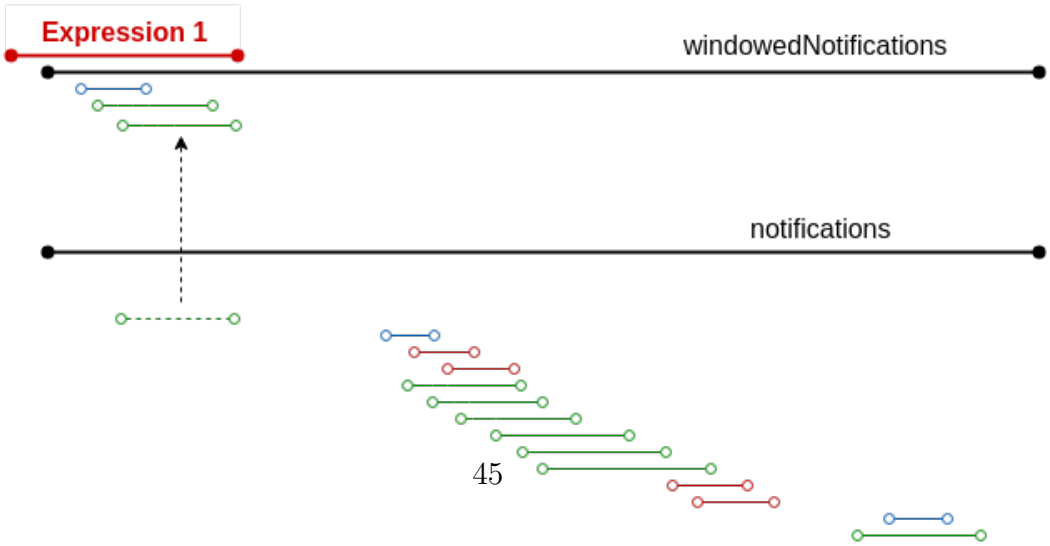
If checkExpression returns false, indicating that the notifications in *windowedNotification* did not trigger *currentExpression*, then another notification from *notifications* is popped off, added to *windowedNotification*, and the process repeats.

To evaluate *currentExpression* against *windowedNotification*, all of the notifications have to fit under the umbrella of the sliding window. That is, the interval of time from the beginning of the earliest notification in *windowedNotification* to the end of the last notification must be less than the window of *currentExpression*. If this interval is larger than the window of *currentExpression*, this indicates that *windowedNotification* has a set of notifications that occurred over a larger interval of time than *currentExpression*'s window allows.

When the interval of *windowedNotification* is greater than the window of *currentExpression*, the queue must be trimmed so until the interval is once again smaller than the window of *currentExpression* and can be evaluated. Trimming *windowedNotification* is done by continually popping off the earliest notification until the interval is smaller than the window of *currentExpression*. Because trimming requires popping the earliest notification, the *windowedNotification* queue is implemented as a priority queue and sorted by earliest start time of the notification, as opposed to the


Expression 1
windowedNotifications
notifications


Expression 1
windowedNotifications
notifications


Expression 1
windowedNotifications
notifications

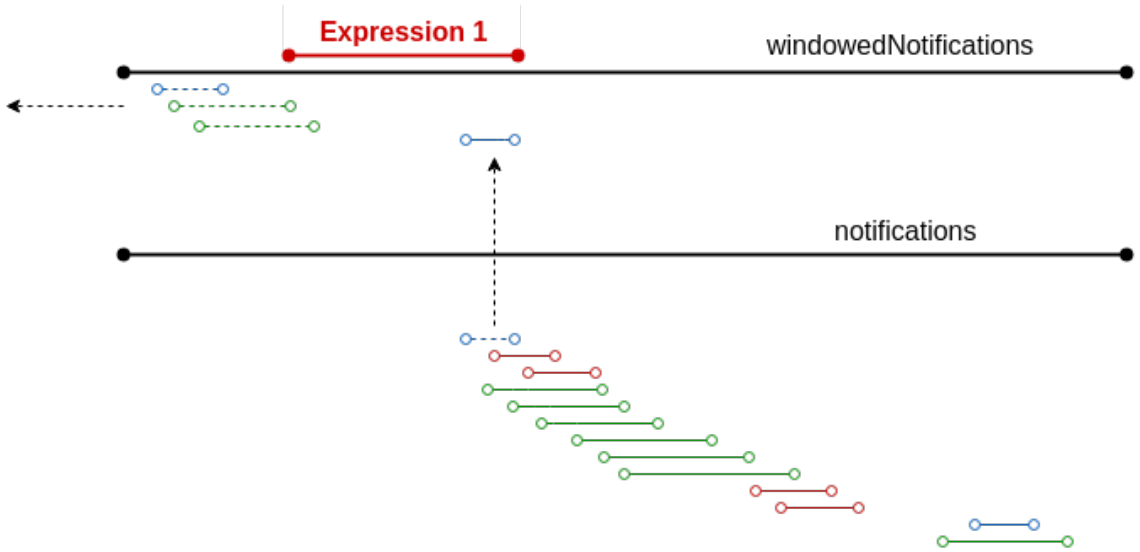


Figure 6.3: Example of windowedNotifications queue popping old notifications.

notifications queue which is sorted by end time.

The *currentExpression* will continue to slide down the timeline, and as it does so the *windowedNotification* will slide along and evaluate against the expression with checkExpression each time a new notification is added. If checkExpresion never returns true, the *currentExpression* will eventually slide to the end of the timeline and the *notifications* queue will run out of notifications. This means that all possible legal combinations of notifications were checked for *currentExpression* and it never evaluated to true. If *currentExpression* does not evaluate to true, then the rule does not either and the processing for that rule is complete. The analyze method iterates to the next rule and checkRule is called again.

If the *currentExpression* does evaluate to true, then the next expression in the rule needs to be evaluated. *currentExpression* is set to the next expression. The *windowedNotification* queue is cleared because all of its notifications were just used in the analysis of the last expression. Additionally, because the next expression must start after the last expression finished, the *notifications* queue is trimmed to the end

of the last expression. That is all notifications in *notifications* that start before the last expression ends are popped off.

This leaves an empty *windowedNotification* queue, a *notifications* queue that has been trimmed to the proper start of the next expression's possible window, and the next expression has been set to *currentExpression*. The window then continues to slide.

If each expression in the rule has been iterated through and evaluates to true, then the rule itself evaluates to true and checkRule returns true.

6.1.2 Checking Expressions and Clauses

The checkExpression and checkClause methods check the conditions for an expression and a clause, respectively, that cause them to evaluate to true.

Recall that for an expression to evaluate to true, just one of its clauses needs evaluate to true. So the checkExpression method iterates through each of its clauses and calls checkClause. If any of the checkClause calls evaluate to true then the checkExpression call evaluates to true. If none of the clauses evaluate to true, then checkExpression evaluates to false.

Recall that for a clause to evaluate to true, all of its monitor points need to have been triggered. That is the checkClause method checks that each one of the monitor points in the clause has a matching notification in the *windowedNotification* queue. If all of the monitor points in the clause have a matching notification in the *windowedNotification* queue then it evaluates to true. If any of the monitor points are missing from *windowedNotification* then it evaluates to false.

6.1.3 Other Methods

The getNotifications method is responsible for creating the queue of notifications used in the sliding window. Each monitor point in the rule uses a list of events to create notifications which are then added to the queue being returned.

The getApplicableEvents method returns a list of all the events related to the rule that are needed to create the notifications. The list of events is sorted by time from oldest to newest. The notifications are created one monitor point at a time by iterating through each of the monitor points in the rule. The process described in Creating Notifications in Design is used to create notifications for each monitor point.

This starts with an event queue, eventQueue, with a capacity equal to the count of the monitor point's threshold. This queue will be used to determine if a monitor point's threshold has been crossed. One at a time events matching the monitor point will be added to eventQueue. When the queue is full, then eventQueue is checked against the monitor point's threshold. If the interval of time from the earliest event in eventQueue to the latest event is less than the threshold's interval, then eventQueue represent a set of events that have crossed the threshold of the monitor point and a notification should be created. The notification spans the interval of the eventQueue. This notification is added to the list notificationsQueue which will eventually be returned.

Whether or not a notification is created, events continue to be added to the queue. When the queue is full this means that to make room the oldest event has to be popped out of the queue. Each time an event is added to the queue when it is full, eventQueue must be checked to see if it triggers the monitor point's threshold. This repeats until there are no longer any events that match the monitor point. Then the next monitor point is iterated to, the eventQueue is cleared, and notifications are created for this monitor point. Each monitor point will parse through the events and

created notifications which are added to notificationsQueue. The notificationsQueue is then sorted by end time and returned.

The getApplicableEvents method creates a list of all the events needed to analyze the rule. To collect these events a query is built that will be used to retrieve the events from the event store. This query will return all events that match any of the monitor points in the rule passed to setRule. Additionally, all of the events must have occurred after the time passed to setEarliest. This time can be one of two values, whichever occurs later. The first option ensures that all of the events occurred within the rule's window, so it is set to rule's window time before *triggerEvent*, that is *triggerEvent*.time - rule.window. The second option is the time the last attack was created for this rule. This is discovered by querying the attack store in the findMostRecentAttackTime method. The events returned from querying the event store are then sorted by time and returned.

Chapter 7

VALIDATION

7.1 Performance Testing

I performed three tests to determine how the performance of the rule engine compared against the performance of the reference engine. The first tested how the rate of adding events impacted performance. The second tested how the number of configured rules or detection points impacted performance. And the third tested how the number of events in the event store impacted performance.

To do this three different AppSensor REST server configurations were used. The first was configured with the reference analysis engine and a single detection point. The second was configured with the rules-based analysis engine and a simple rule built up of a single detection point. The third was configured with the rules-based analysis engine and a complex rule built up of 5 expressions each with a total of 5 monitor points being spread over 3 clauses.

I expected to see the rules-based engine perform worse than the reference engine across all tests, and expected the complex rule configuration to perform worse than the simple configuration across all tests. The results of these tests showed these assumptions to hold true. The simple rule configuration generally performed worse than the reference engine, but not by much. The complex rule configuration, however performed significantly worse than either the simple rule configuration or the reference rules engine.

7.1.1 Test Setup

Each test was set up with a client and a server. The server was AppSensor running in a REST implementation with an in-memory storage configuration. The only thing that would change within the server between tests was the configuration file, which was used to configure the particular rule or detection points needed in that test. The client used was jmeter, an open source network testing tool. The jmeter client was configured in different ways for each test, changing variables such as number of requests sent, time between each request, and the type of event being sent. An example of the jmeter configuration can be seen in Appendix D.

The AppSensor server would run locally and listen for REST messages over port 8085. The jmeter client would run in headless mode and would send requests that add an event to the AppSensor system. Each time the server receives a REST request, it first performs all processing before sending a response. This means that to measure the performance of the different engine configurations, specifically to measure how long it takes for them to process adding an event, the response latency can be used as a confident metric. The longer an engine processes an event, the longer the response will take to return to the client.

The different server configurations for the complex, simple, and reference tests can be seen in Appendices A B C.

All tests were run locally on my Acer Aspire V running Ubuntu 16.04 with an Intel Core i5-6200U CPU @ 2.30GHz 4 and 5.7 GiB of memory. The jmeter client always ran with only a single thread. Each test was run by a bash script that would spawn a new instance of an AppSensor server, start the jmeter run, and then tear down the AppSensor server before starting the next test. This was so that each test was run on a clean, new instance of a server. The script also handled modifying any variables that needed to change for the test at hand.

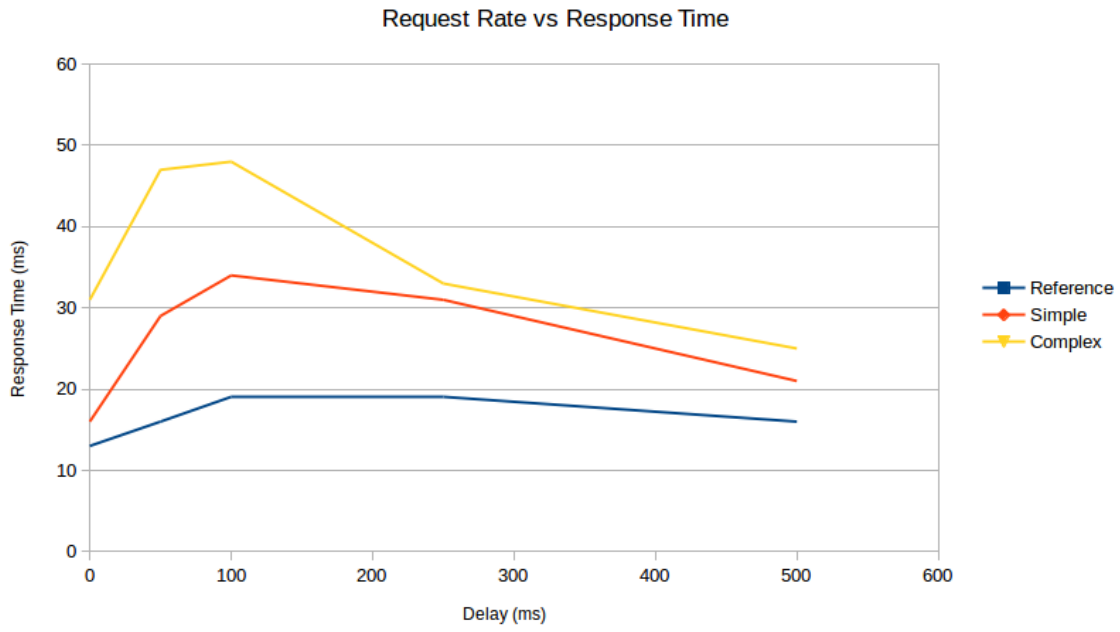


Figure 7.1: Results from test with variable delay between requests.

7.1.2 Test 1 - Rate of Events

This test was designed to see how the rules-based engine held up under heavy load and how the extra processing time might affect responsiveness. To accomplish this the jmeter client added a delay between request that was varied between tests. Delays of 0, 50, 100, 250, and 500 milliseconds were tested, while the number of requests sent was held constant at 200 as was the configuration, which used only one rule or detection point. For each test of 200 requests per delay, the average latency was calculated and graphed which can be seen in Figure 7.1.

As shown in Figure 7.1 as the delay between requests increases, and inversely the rate of incoming request decreases, the latency in requests decreases. This matches the expectations, that the higher the rate of incoming events, the more requests that will be queued up waiting to be processed, which increases the latency in the response. Also as expected, the complex configured rules-based engine was the slowest, followed

by the simple rules-based engine, then the reference engine.

Interestingly, the latency was lower when the requests were being sent out without any delays (the quickest possible rate) before then spiking at 50 and 100 milliseconds. Also worth noting is that the same test configurations of 0 ms delay, 200 requests, and 1 rule/ detection point, were used in another test and resulted in slightly lower response latency. This test was performed from a single computer, and the results may have been more interesting in a network situation with multiple clients sending requests.

7.1.3 Test 2 - Number of Rules / Detection Points

This test was designed to understand the impact of how the number of rules or detection points configured to an AppSensor system impacted the performance of the analysis engines. I expected that as more rules are added to the system, the greater the processing time will be as each applicable rule has to be reviewed. While the number of requests is held constant at 200 and the delay between events is held constant at 0 milliseconds; 1, 2, 4, and 8 configured detection points or rules are tested. For each test the average latency in response time was calculated and graphed as seen in Figure 7.2.

The results demonstrated that I was half right and half wrong. In the case of the complex configured rules-based engine, increasing the number of rules had a large impact on performance. The latency with 8 rules was almost three times as long as with 1 rule. The performance of the simple and reference tests however did not seem to be impacted by the increase - the latency of the test hovering around the amount at each step. That the complex test performed so much worse than the simple test, seems to indicate that the complexity of a rule has a much more significant impact on performance than the number of rules.

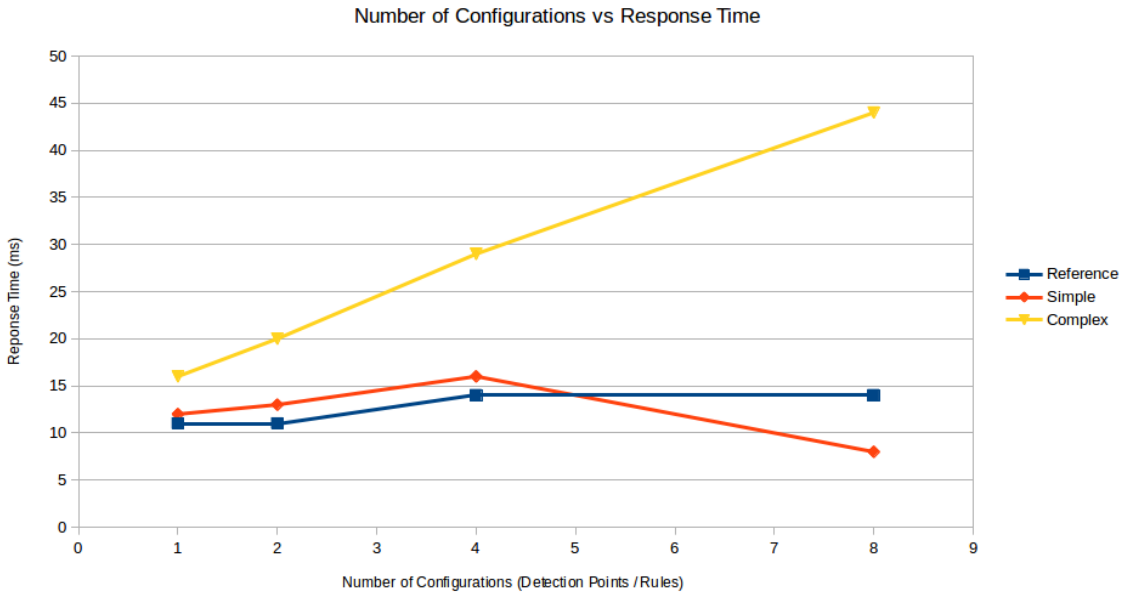


Figure 7.2: Results from test with varied number of configured detection points or rules.

7.1.4 Test 3 - Size of Event Store

The third performance test was left to determine how the number of events in the event store impacted the performance of the engines. Each time an analysis engine runs, it must query for events from the event store. The duration of this query will have an impact on the processing time. This test was run with a 0 ms delay and with a single configured detection point or rule. The test was allowed to run for 5000 events. The results can be seen in Figure 7.3.

The results show that as the number of events in the event store increase, so does the response time. The most glaring result is how much worse the latency is for the complex configuration than either of the other two tests. The results are almost 10 times worse than the reference engine. Both the reference and simple tests will only need to search for events that match a single detection point, whereas the complex test will need to find events that match any of its three different monitor points, making it a more complicated query. While it makes sense that more data in the

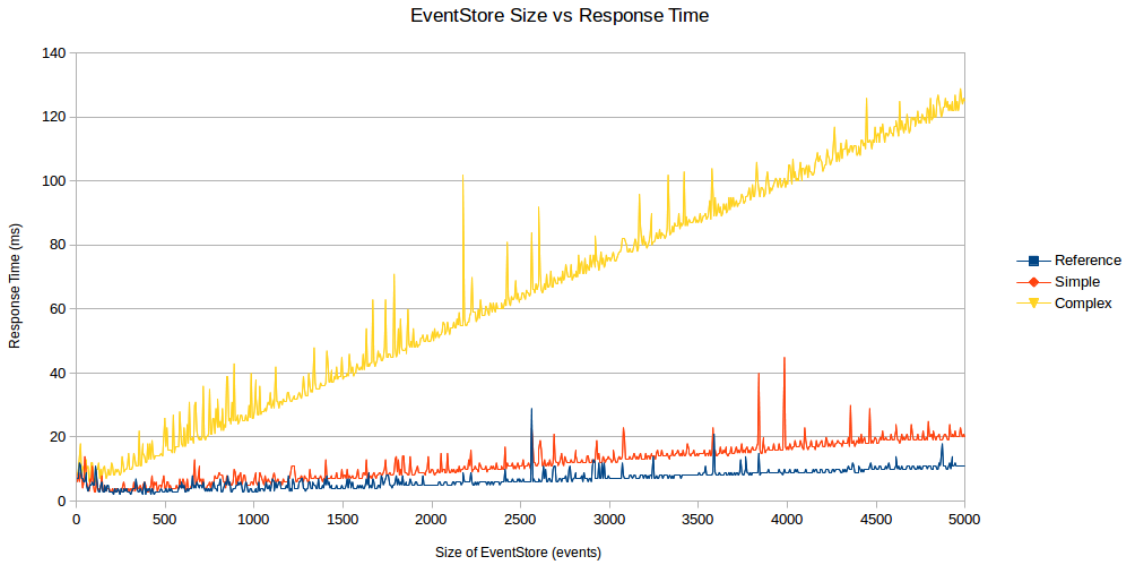


Figure 7.3: Results of test comparing event store size to response time.

data stores would lead to greater query times, the linear relationship between size and latency does not bode well for scale.

Unfortunately, this was a rather flawed test. The event store is implemented with the in-memory storage provider which does not have any optimizations and does not use caching. Had this database implementation been done with a more performant provider the relationship between the number of events and the performance should be logarithmic, as opposed to linear. Additionally, AppSensor is designed so that the number of events that need to be stored at any one time is relatively low. Detection points and rules are intended to be configured over period of time less than 24 hours, so only the last 24 hour worth of data would be needed to retain. These design points mean that the event store will likely never have enough data to warrant realistic concern.

Additionally, each of the configurations are designed to trigger an attack every 5 events, which means that when the engine needs to query for events and uses the last known attack time as a filter, that only 5 events at most are being processed by the

rules engine. A better test would have created a scenario that allowed many more events to be added to the event store without triggering an attack, yet still being related to the rule so that a greater number of events were returned by the query.

7.1.5 Performance Results

As expected, the rules-based engine does not perform as well as the much simpler reference engine. The performance of the rules-based engine does not appear to be a limiting factor, rather already known issues such as query time against large data stores are bigger factors, which are taken into account when designing the system.

7.2 Use Cases

There are a number of realistic scenarios where the use of a rule-based analysis engine has advantages over the single threshold analysis engine. It is not necessary to conduct tests to demonstrate the value of the rule-based engine in these cases, but rather by investigating several different use case example we can see the applications of the rule-based engine. The goal of these use cases will be to highlight where the rules-based engine is a better fit than the threshold analysis engine and as a result provides stronger assurances against false-positives and false-negatives.

7.2.1 Authentication

Most web applications have an authentication page for users to log into. These components are often a focal point for attackers because not only do they provide access to user data and further application functionality, but also because they are often the only public facing page. Monitoring user activity on authentication components can help prevent unwanted probing or bot activity as well as to track unusual

successful logins.

One of the most common authentication defenses is actually an excellent fit for the threshold model: rate limiting. If a user fails to login some amount of times in a minute than an application can lock that user out from attempting to login again for a certain period of time. It is relatively easy to establish a low threshold for failed logins accompanied with an intuitive response to limit attackers.

Many of the other suspicious user metrics associated with authentication, however, are not as easy to attach a single threshold. Unusual characters in the username, password or username too long, and unusual HTTP headers are all indicative of suspicious activity, but are not definitive. Trying to use a single threshold to monitor these type of events would be difficult. A threshold of one event may produce too many false positives as a browser extension may add unusual HTTP headers and unusual characters can be accidently typed.

When combined in a rule, however, there can be stronger confidence in the result. For example, if an authentication request contains unusual characters *and* an unusual HTTP header *and* a username or password was too long, then this user should be blocked and reported. A threshold of one event can be used for each sensor, and false positives can be reduced by aggregating activity from multiple sources.

This same approach can be added to the rate limiting example to make it stronger. If a user fails to login 3 times in a minute then lock them out from logging in for 5 minutes. If, however, a user fails to login 3 times in a minute *and* either a request contained an unusual character *or* an unusual HTTP header *or* a username or password that is too long, then lock the user out indefinitely and report it.

Another approach could be to utilize threat intelligence in combination with sensor input. Perhaps a web application subscribes to threat intelligence that informs them of a set of emails recently leaked in a breach of credentials from another site. It is

possible some of those emails are of users from both sites. If there are multiple failed logins *and* the email matches one in the threat intelligence list then block requests from the destination IP address, as opposed to simply locking the user out for a few minutes. By leveraging data from both the threat intelligence and the failed logins, the rule can recognize malicious activity that could not have been recognized by a single threshold. This helps to reduce false-negatives.

Even if a user successfully logs in there may be indicators that something is not quite right. The geographic location of an IP address or the browser fingerprint of the user might be unusual. For example, if a user that has only ever logged in from California on their iPhone logs in from Russia using Firefox, this would be unusual. May be that person is visiting a friend in Russia and needed to login on their friend's computer, or it may just be an unauthorized user. In this case, if there either one of these unusual things is detected the response could be first to send an email to the user notifying them of a strange login. If both the geographic location *and* the browser fingerprint are unusual then require two factor authentication. If either the geographic location*or* the browser fingerprint are unusual *and* they try to change the password, then require two factor authentication and alert the security team. Rules add extra granularity to automated decision making using AppSensor and can be used to add greater defense in depth.

7.2.2 Fraud

There are many instances where fraud can occur in web applications, but two common instances of it that can occur over a wide range of sites are carding and fake users.

Carding is a process by which criminals will attempt to check if stolen credit card numbers are still working by registering them as a payment method on ecommerce

sites. [21] Often when a user adds a credit card as a payment method, the credit card company will automatically validate whether that card is valid or expired and this will be displayed to the user. Criminals can leverage any ecommerce site then to validate their stolen credit cards.

There are several sensors that would be obvious right away: high rate of requests, multiple credit card numbers used, and multiple failed credit card numbers. All of these would be good indicators of carding attempts and simple threshold would do well for anyone of them. Block an IP if there are 5 requests in a few seconds, if more than 5 different credit card numbers are used in one minute, or if there are 5 failed credit card numbers attempted in a minute. These thresholds have to be set reasonably high, because a user could have bad eyesight and keep punching the wrong credit card number in or a user might have created a new account and want to add all of their numbers at once.

Carding criminals are smart though, and they may only send a few requests to one ecommerce site, then switch to another, only to switch back to the original site again - never triggering the high thresholds that were configured to prevent false positives. If we combine those first three sensors with other indicators we can reduce the thresholds and maintain the same level of confidence. One extra sensor to use would be checking the HTTP method to see if a POST request was submitted without a GET request first being submitted. A POST request should only originate from a user submitting a form on a previous page. A POST request without a prior GET could indicate a carder saving and automating the request to check credit card numbers.

Using this, we could craft a rule that checks if a POST request was sent without a prior GET and the credit card number was invalid, to block that IP address for a period of time. If it occurs again, permanently block it.

Because criminals will often automate their carding, the time between requests is

often very regular which is very uncommon among regular users. Using this we could a rule that checks if the time between requests is exactly the same and multiple credit card number are used, block the IP address for a period of time. If it happens again, blacklist the IP.

Using the rule-based engine we were able to reduce the credit card thresholds that were configured to be high to guard against false-positives by aggregating them with other sensors that, while uncommon, were not definitively indicative of malicious activity without the supporting credit card thresholds.

Another common form of fraud in web applications is in fake, or robot, users. Bot users are common in social media applications where they are often utilized as a source of spam. Applications like Twitter and Facebook are constantly fighting against bots which bother legitimate users. One good indicator of a fake user is the spreading of illegitimate links, which point to sites with low reputation. Legitimate users are also capable of spreading illegitimate links, so as a stand alone metric it can be difficult to detect fake users.

If we combine other bot users tendencies with detection of low reputation links into a rule, however, we can increase confidence in the analysis. If a user has been posting low reputation links *and* has a significantly greater number of friend requests than the average user *and* the shared friends among requested friends is uncommonly low, then this user account should be flagged as a bot and the security team should be notified.

7.2.3 OWASP Top 10

The OWASP Top 10 covers a variety of different web application vulnerabilities. While there are many recommendations on how to fix these vulnerabilities or tools to prevent attacks, there are not a lot of tools that monitor probing of these vul-

nerabilities. Even if a vulnerability does not get exploited, if somebody is prodding around a web application looking for them, this is a problem. Using the rules-based engine, AppSensor can be an effective tool to leverage the OWASP Top 10 to detect suspicious activity.

Knowing that adversaries will probe areas likely to contain a Top 10 vulnerability, we can watch for activity at these points. One common technique attackers use is called fuzzing, where different values are submitted to fields to see if anything interesting or bad happens. When something different does result from fuzzing, this is often a good indication there could be a vulnerability. Often automated, fuzzing can also be done manually, with attackers submitting confusing values to fields.

For example, unfamiliar URLs or invalid parameter names and values are a good indication that an attacker is fuzzing looking for injection or a direct object reference. This could also be an error in entering a URL or a mistyped URL shared with a user. A single threshold for invalid parameter names may be set to 10 in 5 minutes. Probing for XSS often involves submitting special characters to various fields and forms that should not be receiving special characters. We can couple these two very accident prone sensors in a rule to provide more confident results. If 2 invalid parameter names *and* 2 invalid characters detected in form submission, then log user out. If it happens again, lock them out for a period of time and notify the security team.

Probing can eventually lead to exploits and a big target of web applications are databases, where password, credit card information, and other sensitive data is stored. AppSensor configured with rules can be used to detect a data breach. One way to detect a breach is to monitor the database for large files being downloaded. If a file is larger than 500 MB, block that user, log them out and notify the security team. If somebody is downloading huge amounts of data that is a pretty strong indicator that something bad is happening. Depending on the application or the adversary,

however, the threshold for a "large" file may not be as easy to determine. If an application hosts large files or an adversary, trying to stay under the radar, downloads the database in sections rather than all at once, then it may be difficult to set an accurate threshold. Combining the download size with an indicator of what likely came before that, a successful SQli exploit, can improve the accuracy of detection. If a blacklisted SQL character*or* a blacklisted XSS character were submitted anywhere in the application *THEN* a large amount of data was downloaded, then alert the CISO because something bad definitely just happened.

Chapter 8

FUTURE WORK AND CONCLUSION

8.1 Future Work

8.1.1 Complete Integration

The rules-based analysis engine has been added the AppSensor project, but it hasn't been completely integrated. The rules-based engine currently only works in a single configuration: an in-memory storage provider and a local execution mode. Further integration and testing needs will allow the rules-based analysis engine to work with all of the storage providers, execution modes, and integrations. Before rules were introduced to AppSensor it was assumed that an attack could only be triggered by a detection point. So many of the changes required to integrate the new analysis engine involve changing code that leverages that assumption and would otherwise throw an exception when an attack didn't have a detection point to reference, but instead a rule.

These integrations have already been created, but have not yet been fully tested or added to the AppSensor project.

8.1.2 Rule Configuration GUI

Another future endeavor is to create a GUI used to assist the generation of the rule configuration. One of the motivations for this is to allow administrators to use parenthetical operators when writing their rule logic, which isn't supported in the rules model. And although they aren't supported, the same logic can always be achieved using the distributive property.

So to create a rule for the logic "Sensor1 and (Sensor2 or Sensor3)" It would have to be configured as "Sensor1 and Sensor2 or Sensor1 and Sensor 3".

The tool would allow administrators to write the rule logic in a user-friendly way that allows them to better understand, control, and manage the rules they want to create. The logic created in the GUI could then be used to generate the proper XML that is read in by AppSensor and generates the rules for the analysis engine.

8.1.3 Notification Data Store

The most difficult and interesting future work is to explore the performance gains by adding a data store for notifications. Each time a new event is added to the system and a rule is checked, it performs very time intensive, redundant work in recreating all the notifications from the related events. This redundancy could be cut out by adding another data store and engine for notifications.

Instead of checking all of the rules each time an event is created, the event analysis engine would actually revert back to the same logic as the reference engine. Instead of checking detection points and creating attacks, however, it would check monitor points and create notifications. These notifications would be added to the notifications store which would have a matching analysis engine. The notifications analysis engine would run its analyze method each time a notification was created, and would then check each of the rules using the notifications in the notification store.

This would remove the redundant work done by the getNotifications method each time a rule is checked. Because the order of magnitude for each of the key structures used in the rule analysis can vary drastically between different implementations, it's difficult to even estimate how much this change would improve the performance of the engine. However, if we characterize the currently implementation as roughly O (# of rules x # of monitor points x # of events) then the improved implementation

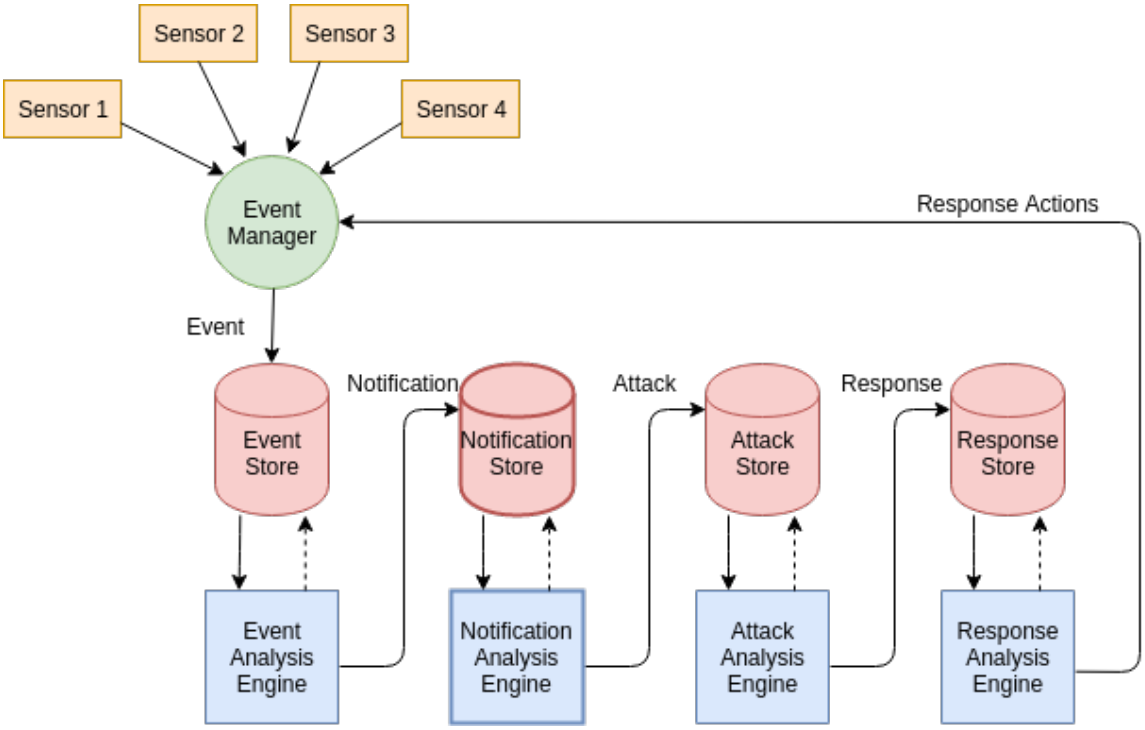


Figure 8.1: Results from test with varied number of configured detection points or rules.

should be roughly O (# of rules x # notifications) where events > > monitor points or rule and events > notifications. The problem is that it is difficult to determine how much larger the magnitude of events is than the magnitude of the notifications. It depends on factors like how many sensors an implementation is configured with, how those sensors are tuned, and how the thresholds of the monitor points are set.

Creating these changes and evaluating the resulting improvements on various configurations would be useful experimentation. If the improvements are drastic enough, it could be worth the extras complexity of adding an extra analysis engine and data store.

8.2 Conclusion

The goal of this thesis was to improve the state of web application security by contributing to the development of application level IDS through the AppSensor project. The rules-based engine was built to increase the flexibility of configuration and improve the accuracy of analysis of the AppSensor system. After writing the rules-based analysis engines and the support code I explored common use cases that demonstrated how the rules-based engine could be leveraged to be more accurate. These use cases showed that it could be utilized to lower thresholds and thus decrease the chances for false-negatives or conversely because more than one sensor are factoring into a decision that there can be greater confidence in the result and a greater likelihood that false positives goes down as well. Performance tests showed that although it performed worse than the reference engine, it still was able to process events within a realistic amount of time. Further work needs to be done to finish integration of the engine into all components of the AppSensor system and investigation into an extra data store could provide insight to improving performance. The rules-based engine has been added to the AppSensor project and is available via GitHub.

BIBLIOGRAPHY

[1] Application defender. https://saas.hpe.com/en-us/software/application-defender.

[2] Prevoty runtime application security. https://www.prevoty.com/science/rasp.

[3] Repsheet - threat intelligence toolkit. https://getrepsheet.com.

[4] Runtime application self-protection contrast protect. https://www.contrastsecurity.com/runtime-application-self-protection-rasp.

[5] Runtime application self-protection (rasp). http://www.waratek.com/runtime-application-self-protection-rasp/.

[6] *Salesforce Security Guide: Transaction Security Policies*.

[7] *The true cost of a Web Application attack*. leaseweb.

[8] User guide. http://appsensor.org/user-guide.html.

[9] *State of Web Application Security*. Feb 2011.

[10] A short history of javascript. https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript, Jun 2012.

[11] Owasp top ten project. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, 2013.

[12] *2015 Cost of Data Breach Study: Global Analysis*. May 2015.

[13] *2015 Web Application Attack Report (WAAR)*. Imperva, 6 edition, 2015.

[14] *Alert Logic Cloud Security Report*. 2015.

[15] Appsensor detectionpoints. `https://www.owasp.org/index.php/AppSensor_DetectionPoints`, Jul 2015.

[16] *Verizons 2015 Data Breach Investigations Repor (DBIR)*. Verizon, 1 edition, 2015.

[17] *Verizons 2016 Data Breach Investigations Repor (DBIR)*. Verizon, 2016.

[18] Vulnerability definition. `https://www.owasp.org/index.php/Category:Vulnerability`, Jun 2016.

[19] Arxan. Rasp. `https://www.arxan.com/technology/rasp/`.

[20] P. Asadoorian. *Getting a Grasp on Rasp*.

[21] A. Bedra. Adaptive security. YOW! 2015, 2015.

[22] J. Feiman. *Maverick* Research: Stop Protecting Your Apps; It's Time for Apps to Protect Themselves*. Sep 2014.

[23] B. Goldfarb. Introducing real-time, automated security policies with salesforce shield. `https://www.salesforce.com/blog/2016/03/introducing-automated-security-policies-salesforce-shield.html`, Mar 2016.

[24] A. Infante. Why flash needs to die (and how you can get rid of it), Sep 2015.

[25] K. Ingham and S. Forrest. *A History and Survey of Network Firewalls*. 2002.

[26] K. Ingham and S. Forrest. *Network Firewalls*. University of New Mexico, 1 edition, 2005.

[27] B. Krebs. Home depot hit by same malware as target, Sep 2014.

[28] E. Lebanidze. *Securing Enterprise Web Applications at the Source: An Application Security Perspective*. Jun 2006.

[29] X. Li and Y. Xue. A survey on web application security. 2011.

[30] S. C. Liang. Understanding behavioural detection of antivirus, Apr 2016.

[31] G. R. Malan, D. Watson, F. Jahanian, and P. Howell. *Transport and Application Protocol Scrubbing*. 2000.

[32] A. W. Mathews. Anthem: Hacked database included 78.8 million people. *The Wall Street Journal*, Feb 2015.

[33] J. Meier, A. Mackman, M. Dunner, S. Vasireddy, R. Escamilla, and A. Murukan. *Improving Web Application Security: Threats and Countermeasures Roadmap*. Jun 2003.

[34] J. Murdock. Did attackers use an 'sql injection' to hack into the qatar national bank?, Apr 2016.

[35] T. Ormandy. *Sophail: Applied attacks against Sophos Antivirus*.

[36] T. Ormandy. How to compromise the enterprise endpoint, Jun 2016.

[37] T. Ormandy. Security software certification, Mar 2016.

[38] Owasp.org. Owasp appsensor project - owasp, 2015.

[39] J. Pagliery. Adobe has an epically abysmal security record. *CNN*, Oct 2013.

[40] J. Silver-Greenberg, M. Goldstein, and N. Perlroth. Jpmorgan chase hacking affects 76 million households. *The New York Times*, Oct 2014.

[41] D. Watson, M. Smart, G. R. Malan, and F. Jahanian. Protocol scrubbing: Network security through transparent flow modification. *IEEE/ACM Transactions on Networking*, 12(2):261273, Apr 2004.

APPENDICES

Appendix A

‘REFERENCE’ TESTING CONFIGURATION

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <appsensor-server-config xmlns="https://www.owasp.org/index.php/OWASP_AppSensor_Project/xsd/
       appsensor_server_config_2.0.xsd">
3
4    <client-application-identification-header-name>X-Appsensor-Client-Application-Name2</client-application-
         identification-header-name>
5
6    <geolocation enabled="true" databasePath="src/main/resources/GeoLite2-City.mmdb" />
7
8    <client-applications>
9      <client-application>
10       <name>myclientapp</name>
11       <roles>
12         <role>ADD_EVENT</role>
13         <role>ADD_ATTACK</role>
14         <role>GET_RESPONSES</role>
15         <role>EXECUTE_REPORT</role>
16       </roles>
17     </client-application>
18     <client-application>
19       <name>myclientgeoapp1</name>
20       <roles>
21         <role>ADD_EVENT</role>
22         <role>ADD_ATTACK</role>
23         <role>GET_RESPONSES</role>
24         <role>EXECUTE_REPORT</role>
25       </roles>
26       <!-- ireland -->
27       <ip-address latitude="52.629678" longitude="-7.873585">10.10.10.5</ip-address>
28     </client-application>
29     <client-application>
30       <name>myclientgeoapp2</name>
31       <roles>
32         <role>ADD_EVENT</role>
33         <role>ADD_ATTACK</role>
34         <role>GET_RESPONSES</role>
35         <role>EXECUTE_REPORT</role>
36       </roles>
37       <!-- brazil -->
38       <ip-address latitude="-7.471493" longitude="-47.248578">10.10.10.6</ip-address>
39     </client-application>
40     <client-application>
41       <name>myclientgeoapp3</name>
42       <roles>
```

```
43          <role>ADD_EVENT</role>
44          <role>ADD_ATTACK</role>
45          <role>GET_RESPONSES</role>
46          <role>EXECUTE_REPORT</role>
47        </roles>
48        <!-- russia -->
49        <ip-address latitude="59.164625" longitude="123.96234">10.10.10.7</ip-address>
50      </client-application>
51      <client-application>
52        <name>myclientgeoapp4</name>
53        <roles>
54          <role>ADD_EVENT</role>
55          <role>ADD_ATTACK</role>
56          <role>GET_RESPONSES</role>
57          <role>EXECUTE_REPORT</role>
58        </roles>
59        <!-- india -->
60        <ip-address latitude="12.875989" longitude="77.556100">10.10.10.8</ip-address>
61      </client-application>
62    </client-applications>
63
64    <correlation-config>
65      <correlated-client-set>
66        <client-application-name>A</client-application-name>
67        <client-application-name>B</client-application-name>
68        <client-application-name>C</client-application-name>
69      </correlated-client-set>
70      <correlated-client-set>
71        <client-application-name>X</client-application-name>
72        <client-application-name>Y</client-application-name>
73        <client-application-name>Z</client-application-name>
74      </correlated-client-set>
75    </correlation-config>
76
77    <detection-points>
78      <detection-point>
79        <category>Input Validation</category>
80        <id>IE1</id>
81        <threshold>
82          <count>5</count>
83          <interval unit="seconds">30</interval>
84        </threshold>
85        <responses>
86          <response>
87            <action>log</action>
88            <interval unit="minutes">10</interval>
89          </response>
90          <response>
91            <action>logout</action>
92          </response>
93        </responses>
94      </detection-point>
95    </detection-points>
96  </appsensor-server-config>
```

Appendix B

‘SIMPLE’ TESTING CONFIGURATION

```
<?xml version="1.0" encoding="UTF-8"?>
<appsensor-server-config xmlns="https://www.owasp.org/index.php/OWASP_AppSensor_Project/xsd/
     appsensor_server_config_2.0.xsd">

  <client-application-identification-header-name>X-Appsensor-Client-Application-Names</client-application-
       identification-header-name>

  <geolocation enabled="true" databasePath="src/main/resources/GeoLite2-City.mmdb" />

  <client-applications>
    <client-application>
      <name>myclientapp</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
      </roles>
    </client-application>
    <client-application>
      <name>myclientgeoapp1</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
      </roles>
      <!-- ireland -->
      <ip-address latitude="52.629678" longitude="-7.873585">10.10.10.5</ip-address>
    </client-application>
    <client-application>
      <name>myclientgeoapp2</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
      </roles>
      <!-- brazil -->
      <ip-address latitude="-7.471493" longitude="-47.248578">10.10.10.6</ip-address>
    </client-application>
    <client-application>
      <name>myclientgeoapp3</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
```

```
      </roles>
      <!-- russia -->
      <ip-address latitude="59.164625" longitude="123.96234">10.10.10.7</ip-address>
    </client-application>
    <client-application>
      <name>myclientgeoapp4</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
      </roles>
      <!-- india -->
      <ip-address latitude="12.875989" longitude="77.556100">10.10.10.8</ip-address>
    </client-application>
  </client-applications>

  <correlation-config>
    <correlated-client-set>
      <client-application-name>A</client-application-name>
      <client-application-name>B</client-application-name>
      <client-application-name>C</client-application-name>
    </correlated-client-set>
    <correlated-client-set>
      <client-application-name>X</client-application-name>
      <client-application-name>Y</client-application-name>
      <client-application-name>Z</client-application-name>
    </correlated-client-set>
  </correlation-config>

  <rules>
    <rule guid="00000000-0000-0000-0000-000000000005">
      <name>Simple Rule</name>
      <window unit="seconds">30</window>
      <expressions>
        <expression>
          <window unit="seconds">30</window>
          <clauses>
            <clause>
              <monitor-points>
                <monitor-point guid="00000000-0000-0000-0000-000000000006">
                  <category>Input Validation</category>
                  <id>IE1</id>
                  <threshold>
                    <count>5</count>
                    <interval unit="seconds">30</interval>
                  </threshold>
                </monitor-point>
              </monitor-points>
            </clause>
          </clauses>
        </expression>
      </expressions>

      <responses>
```

```
102           <response>
103             <action>log</action>
104           </response>
105         </responses>
106       </rule>
107     </rules>
108
109   </appsensor-server-config>
```

Appendix C

‘COMPLEX’ TESTING CONFIGURATION

```
<?xml version="1.0" encoding="UTF-8"?>
<appsensor-server-config xmlns="https://www.owasp.org/index.php/OWASP_AppSensor_Project/xsd/
     appsensor_server_config_2.0.xsd">

  <client-application-identification-header-name>X-Appsensor-Client-Application-Name2</client-application-
       identification-header-name>

  <geolocation enabled="true" databasePath="src/main/resources/GeoLite2-City.mmdb" />

  <client-applications>
    <client-application>
      <name>myclientapp</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
      </roles>
    </client-application>
    <client-application>
      <name>myclientgeoapp1</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
      </roles>
      <!-- ireland -->
      <ip-address latitude="52.629678" longitude="-7.873585">10.10.10.5</ip-address>
    </client-application>
    <client-application>
      <name>myclientgeoapp2</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
      </roles>
      <!-- brazil -->
      <ip-address latitude="-7.471493" longitude="-47.248578">10.10.10.6</ip-address>
    </client-application>
    <client-application>
      <name>myclientgeoapp3</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
```

```
      </roles>
      <!-- russia -->
      <ip-address latitude="59.164625" longitude="123.96234">10.10.10.7</ip-address>
    </client-application>
    <client-application>
      <name>myclientgeoapp4</name>
      <roles>
        <role>ADD_EVENT</role>
        <role>ADD_ATTACK</role>
        <role>GET_RESPONSES</role>
        <role>EXECUTE_REPORT</role>
      </roles>
      <!-- india -->
      <ip-address latitude="12.875989" longitude="77.556100">10.10.10.8</ip-address>
    </client-application>
  </client-applications>

  <correlation-config>
    <correlated-client-set>
      <client-application-name>A</client-application-name>
      <client-application-name>B</client-application-name>
      <client-application-name>C</client-application-name>
    </correlated-client-set>
    <correlated-client-set>
      <client-application-name>X</client-application-name>
      <client-application-name>Y</client-application-name>
      <client-application-name>Z</client-application-name>
    </correlated-client-set>
  </correlation-config>

  <rules>
    <rule guid="00000000-0000-0000-0000-000000000000">
      <name>Rule 1</name>
      <window unit="minutes">3</window>
      <expressions>
        <expression>
          <window unit="seconds">30</window>
          <clauses>
            <clause>
              <monitor-points>
                <monitor-point guid="00000000-0000-0000-0000-000000000001">
                  <category>Input Validation</category>
                  <id>IE1</id>
                  <threshold>
                    <count>50</count>
                    <interval unit="seconds">30</interval>
                  </threshold>
                </monitor-point>

                <monitor-point guid="00000000-0000-0000-0000-000000000002">
                  <category>Input Validation</category>
                  <id>IE2</id>
                  <threshold>
                    <count>1</count>
                    <interval unit="seconds">10</interval>
```

```
                </threshold>
              </monitor-point>
            </monitor-points>
          </clause>

          <clause>
            <monitor-points>
              <monitor-point guid="00000000-0000-0000-0000-000000000003">
                <category>Input Validation</category>
                <id>RE3</id>
                <threshold>
                  <count>2</count>
                  <interval unit="seconds">20</interval>
                </threshold>
              </monitor-point>

              <monitor-point guid="00000000-0000-0000-0000-000000000012">
                <category>Input Validation</category>
                <id>IE2</id>
                <threshold>
                  <count>1</count>
                  <interval unit="seconds">10</interval>
                </threshold>
              </monitor-point>
            </monitor-points>
          </clause>

          <clause>
            <monitor-points>
              <monitor-point guid="00000000-0000-0000-0000-000000000022">
                <category>Input Validation</category>
                <id>IE2</id>
                <threshold>
                  <count>1</count>
                  <interval unit="seconds">10</interval>
                </threshold>
              </monitor-point>
            </monitor-points>
          </clause>
        </clauses>
      </expression>

      <expression>
        <window unit="seconds">30</window>
        <clauses>
          <clause>
            <monitor-points>
              <monitor-point guid="00000000-0000-0000-0000-000000000031">
                <category>Input Validation</category>
                <id>IE1</id>
                <threshold>
                  <count>50</count>
                  <interval unit="seconds">30</interval>
                </threshold>
              </monitor-point>
```

```
157
158                 <monitor-point guid="00000000-0000-0000-0000-000000000032">
159                   <category>Input Validation</category>
160                   <id>IE2</id>
161                   <threshold>
162                     <count>1</count>
163                     <interval unit="seconds">10</interval>
164                   </threshold>
165                 </monitor-point>
166               </monitor-points>
167             </clause>
168
169             <clause>
170               <monitor-points>
171                 <monitor-point guid="00000000-0000-0000-0000-000000000033">
172                   <category>Input Validation</category>
173                   <id>RE3</id>
174                   <threshold>
175                     <count>2</count>
176                     <interval unit="seconds">20</interval>
177                   </threshold>
178                 </monitor-point>
179
180                 <monitor-point guid="00000000-0000-0000-0000-000000000042">
181                   <category>Input Validation</category>
182                   <id>IE2</id>
183                   <threshold>
184                     <count>1</count>
185                     <interval unit="seconds">10</interval>
186                   </threshold>
187                 </monitor-point>
188               </monitor-points>
189             </clause>
190
191             <clause>
192               <monitor-points>
193                 <monitor-point guid="00000000-0000-0000-0000-000000000052">
194                   <category>Input Validation</category>
195                   <id>IE2</id>
196                   <threshold>
197                     <count>1</count>
198                     <interval unit="seconds">10</interval>
199                   </threshold>
200                 </monitor-point>
201               </monitor-points>
202             </clause>
203           </clauses>
204         </expression>
205
206         <expression>
207           <window unit="seconds">30</window>
208           <clauses>
209             <clause>
210               <monitor-points>
211                 <monitor-point guid="00000000-0000-0000-0000-000000000021">
```

```
212                   <category>Input Validation</category>
213                   <id>IE1</id>
214                   <threshold>
215                     <count>50</count>
216                     <interval unit="seconds">30</interval>
217                   </threshold>
218                 </monitor-point>
219
220                 <monitor-point guid="00000000-0000-0000-0000-000000000072">
221                   <category>Input Validation</category>
222                   <id>IE2</id>
223                   <threshold>
224                     <count>1</count>
225                     <interval unit="seconds">10</interval>
226                   </threshold>
227                 </monitor-point>
228               </monitor-points>
229             </clause>
230
231             <clause>
232               <monitor-points>
233                 <monitor-point guid="00000000-0000-0000-0000-000000000023">
234                   <category>Input Validation</category>
235                   <id>RE3</id>
236                   <threshold>
237                     <count>2</count>
238                     <interval unit="seconds">20</interval>
239                   </threshold>
240                 </monitor-point>
241
242                 <monitor-point guid="00000000-0000-0000-0000-000000000082">
243                   <category>Input Validation</category>
244                   <id>IE2</id>
245                   <threshold>
246                     <count>1</count>
247                     <interval unit="seconds">10</interval>
248                   </threshold>
249                 </monitor-point>
250               </monitor-points>
251             </clause>
252
253             <clause>
254               <monitor-points>
255                 <monitor-point guid="00000000-0000-0000-0000-000000000092">
256                   <category>Input Validation</category>
257                   <id>IE2</id>
258                   <threshold>
259                     <count>1</count>
260                     <interval unit="seconds">10</interval>
261                   </threshold>
262                 </monitor-point>
263               </monitor-points>
264             </clause>
265           </clauses>
266         </expression>
```

```
267
268         <expression>
269           <window unit="seconds">30</window>
270           <clauses>
271             <clause>
272               <monitor-points>
273                 <monitor-point guid="00000000-0000-0000-0000-000000000041">
274                   <category>Input Validation</category>
275                   <id>IE1</id>
276                   <threshold>
277                     <count>50</count>
278                     <interval unit="seconds">30</interval>
279                   </threshold>
280                 </monitor-point>
281
282                 <monitor-point guid="00000000-0000-0000-0000-000000000102">
283                   <category>Input Validation</category>
284                   <id>IE2</id>
285                   <threshold>
286                     <count>1</count>
287                     <interval unit="seconds">10</interval>
288                   </threshold>
289                 </monitor-point>
290               </monitor-points>
291             </clause>
292
293             <clause>
294               <monitor-points>
295                 <monitor-point guid="00000000-0000-0000-0000-000000000043">
296                   <category>Input Validation</category>
297                   <id>RE3</id>
298                   <threshold>
299                     <count>2</count>
300                     <interval unit="seconds">20</interval>
301                   </threshold>
302                 </monitor-point>
303
304                 <monitor-point guid="00000000-0000-0000-0000-000000000112">
305                   <category>Input Validation</category>
306                   <id>IE2</id>
307                   <threshold>
308                     <count>1</count>
309                     <interval unit="seconds">10</interval>
310                   </threshold>
311                 </monitor-point>
312               </monitor-points>
313             </clause>
314
315             <clause>
316               <monitor-points>
317                 <monitor-point guid="00000000-0000-0000-0000-000000000122">
318                   <category>Input Validation</category>
319                   <id>IE2</id>
320                   <threshold>
321                     <count>1</count>
```

```
                    <interval unit="seconds">10</interval>
                  </threshold>
                </monitor-point>
              </monitor-points>
            </clause>
          </clauses>
        </expression>

        <expression>
          <window unit="seconds">30</window>
          <clauses>
            <clause>
              <monitor-points>
                <monitor-point guid="00000000-0000-0000-0000-000000000051">
                  <category>Input Validation</category>
                  <id>IE1</id>
                  <threshold>
                    <count>50</count>
                    <interval unit="seconds">30</interval>
                  </threshold>
                </monitor-point>

                <monitor-point guid="00000000-0000-0000-0000-000000000132">
                  <category>Input Validation</category>
                  <id>IE2</id>
                  <threshold>
                    <count>1</count>
                    <interval unit="seconds">10</interval>
                  </threshold>
                </monitor-point>
              </monitor-points>
            </clause>

            <clause>
              <monitor-points>
                <monitor-point guid="00000000-0000-0000-0000-000000000053">
                  <category>Input Validation</category>
                  <id>RE3</id>
                  <threshold>
                    <count>2</count>
                    <interval unit="seconds">20</interval>
                  </threshold>
                </monitor-point>

                <monitor-point guid="00000000-0000-0000-0000-000000000142">
                  <category>Input Validation</category>
                  <id>IE2</id>
                  <threshold>
                    <count>1</count>
                    <interval unit="seconds">10</interval>
                  </threshold>
                </monitor-point>
              </monitor-points>
            </clause>

```

```
            <clause>
              <monitor-points>
                <monitor-point guid="00000000-0000-0000-0000-000000000152">
                  <category>Input Validation</category>
                  <id>IE2</id>
                  <threshold>
                    <count>1</count>
                    <interval unit="seconds">10</interval>
                  </threshold>
                </monitor-point>
              </monitor-points>
            </clause>
          </clauses>
        </expression>
      </expressions>

      <responses>
        <response>
          <action>log</action>
        </response>

        <response>
          <action>logout</action>
        </response>

        <response>
          <action>disableUser</action>
        </response>

        <response>
          <action>disableComponentForSpecificUser</action>
          <interval unit="minutes">30</interval>
        </response>

        <response>
          <action>disableComponentForAllUsers</action>
          <interval unit="minutes">30</interval>
        </response>
      </responses>
    </rule>
  </rules>

</appsensor-server-config>
```

Appendix D

JMETER TESTING CONFIGURATION

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="3.1" jmeter="3.1 r1770033">
 <hashTree>
  <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="LoadTest" enabled="true">
   <stringProp name="TestPlan.comments"></stringProp>
   <boolProp name="TestPlan.functional_mode">false</boolProp>
   <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
   <elementProp name="TestPlan.user_defined_variables" elementType="Arguments" guiclass="ArgumentsPanel"
   testclass="Arguments" testname="User Defined Variables" enabled="true">
    <collectionProp name="Arguments.arguments"/>
   </elementProp>
   <stringProp name="TestPlan.user_define_classpath"></stringProp>
  </TestPlan>
  <hashTree>
   <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Thread Group" enabled="true">
    <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
    <elementProp name="ThreadGroup.main_controller" elementType="LoopController" guiclass="LoopControlPanel"
    testclass="LoopController" testname="Loop Controller" enabled="true">
     <boolProp name="LoopController.continue_forever">false</boolProp>
     <stringProp name="LoopController.loops">200</stringProp>
    </elementProp>
    <stringProp name="ThreadGroup.num_threads">1</stringProp>
    <stringProp name="ThreadGroup.ramp_time">1</stringProp>
    <longProp name="ThreadGroup.start_time">1488868485000</longProp>
    <longProp name="ThreadGroup.end_time">1488868485000</longProp>
    <boolProp name="ThreadGroup.scheduler">false</boolProp>
    <stringProp name="ThreadGroup.duration">0.1</stringProp>
    <stringProp name="ThreadGroup.delay"></stringProp>
   </ThreadGroup>
   <hashTree>
    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add Event"
    enabled="true">
     <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
     <elementProp name="HTTPsampler.Arguments" elementType="Arguments">
      <collectionProp name="Arguments.arguments">
       <elementProp name="" elementType="HTTPArgument">
        <boolProp name="HTTPArgument.always_encode">false</boolProp>
        <stringProp name="Argument.value">{&quot;user&quot;:{&quot;username&quot;:&quot;
        bob&quot;,&quot;ipAddress&quot;:{&quot;address&quot;:&quot;10.10.10.1&quot;,&quot
        ;geoLocation&quot;:{&quot;latitude&quot;:37.596758,&quot;longitude&quot;:-121.6479
        92}}},&quot;detectionPoint&quot;:{&quot;category&quot;:&quot;Input Validation&quot;,&quot;
        label&quot;:&quot;IE1&quot;,&quot;responses&quot;:[]},&quot;timestamp&
        quot;:&quot;${dt1}&quot;,&quot;detectionSystem&quot;:{&quot;detectionSystemId&quot;:&quot;
        myclientapp&quot;},&quot;metadata&quot;:[]}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
       </elementProp>
      </collectionProp>
     </elementProp>
```

```
      <stringProp name="HTTPSampler.domain">localhost</stringProp>
      <stringProp name="HTTPSampler.port">8085</stringProp>
      <stringProp name="HTTPSampler.connect_timeout"></stringProp>
      <stringProp name="HTTPSampler.response_timeout"></stringProp>
      <stringProp name="HTTPSampler.protocol"></stringProp>
      <stringProp name="HTTPSampler.contentEncoding"></stringProp>
      <stringProp name="HTTPSampler.path">api/v1.0/events</stringProp>
      <stringProp name="HTTPSampler.method">POST</stringProp>
      <boolProp name="HTTPSampler.follow_redirects">false</boolProp>
      <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
      <boolProp name="HTTPSampler.use_keepalive">false</boolProp>
      <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
      <stringProp name="HTTPSampler.implementation">Java</stringProp>
      <boolProp name="HTTPSampler.monitor">false</boolProp>
      <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
     </HTTPSamplerProxy>
     <hashTree>
      <HeaderManager guiclass="HeaderPanel" testclass="HeaderManager" testname="AppSensor &amp;
      ContentType Headers" enabled="true">
       <collectionProp name="HeaderManager.headers">
        <elementProp name="" elementType="Header">
         <stringProp name="Header.name">X-Appsensor-Client-Application-Name2</stringProp>
         <stringProp name="Header.value">myclientapp</stringProp>
        </elementProp>
        <elementProp name="" elementType="Header">
         <stringProp name="Header.name">Content-Type</stringProp>
         <stringProp name="Header.value">application/json</stringProp>
        </elementProp>
       </collectionProp>
      </HeaderManager>
      <hashTree/>
      <JSR223PreProcessor guiclass="TestBeanGUI" testclass="JSR223PreProcessor"
      testname="JSR223 PreProcessor" enabled="true">
       <stringProp name="cacheKey"></stringProp>
       <stringProp name="filename"></stringProp>
       <stringProp name="parameters"></stringProp>
       <stringProp name="script">import java.text.SimpleDateFormat;
import java.util.TimeZone;
import java.util.Date;

Date now = new Date();

SimpleDateFormat gmtFormat = new SimpleDateFormat(&quot;yyyy-MM-dd&apos;T&apos;HH:mm:ss.SSS&apos;Z&apos;&quot;);
TimeZone gmtTime = TimeZone.getTimeZone(&quot;GMT&quot;);
gmtFormat.setTimeZone(gmtTime);

//String endDate = vars.get(&quot;endDate&quot;);
//Date endDateAsDate= estFormat.parse(endDate);

vars.put(&quot;dt1&quot;, gmtFormat.format(now));

//${__time(yyyy-MM-dd&apos;T&apos;HH:mm:ss.SSS&apos;Z&apos;)}</stringProp>
       <stringProp name="scriptLanguage">groovy</stringProp>
      </JSR223PreProcessor>
      <hashTree/>
```

```
104       <ResultCollector guiclass="ViewResultsFullVisualizer" testclass="ResultCollector"
105       testname="View Results Tree" enabled="true">
106        <boolProp name="ResultCollector.error_logging">false</boolProp>
107        <objProp>
108         <name>saveConfig</name>
109         <value class="SampleSaveConfiguration">
110          <time>true</time>
111          <latency>true</latency>
112          <timestamp>true</timestamp>
113          <success>true</success>
114          <label>true</label>
115          <code>true</code>
116          <message>true</message>
117          <threadName>true</threadName>
118          <dataType>true</dataType>
119          <encoding>false</encoding>
120          <assertions>true</assertions>
121          <subresults>true</subresults>
122          <responseData>false</responseData>
123          <samplerData>false</samplerData>
124          <xml>false</xml>
125          <fieldNames>true</fieldNames>
126          <responseHeaders>false</responseHeaders>
127          <requestHeaders>false</requestHeaders>
128          <responseDataOnError>false</responseDataOnError>
129          <saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMessage>
130          <assertionsResultsToSave>0</assertionsResultsToSave>
131          <bytes>true</bytes>
132          <sentBytes>true</sentBytes>
133          <threadCounts>true</threadCounts>
134          <idleTime>true</idleTime>
135          <connectTime>true</connectTime>
136         </value>
137        </objProp>
138        <stringProp name="filename"></stringProp>
139       </ResultCollector>
140       <hashTree/>
141      </hashTree>
142      <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Report Attacks"
143      enabled="true">
144       <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel"
145       testclass="Arguments" testname="User Defined Variables" enabled="true">
146        <collectionProp name="Arguments.arguments"/>
147       </elementProp>
148       <stringProp name="HTTPSampler.domain">localhost</stringProp>
149       <stringProp name="HTTPSampler.port">8085</stringProp>
150       <stringProp name="HTTPSampler.connect_timeout"></stringProp>
151       <stringProp name="HTTPSampler.response_timeout"></stringProp>
152       <stringProp name="HTTPSampler.protocol"></stringProp>
153       <stringProp name="HTTPSampler.contentEncoding"></stringProp>
154       <stringProp name="HTTPSampler.path">/api/v1.0/reports/attacks</stringProp>
155       <stringProp name="HTTPSampler.method">GET</stringProp>
156       <boolProp name="HTTPSampler.follow_redirects">false</boolProp>
157       <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
158       <boolProp name="HTTPSampler.use_keepalive">false</boolProp>
```

```
159       <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
160       <stringProp name="HTTPSampler.implementation">Java</stringProp>
161       <boolProp name="HTTPSampler.monitor">false</boolProp>
162       <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
163      </HTTPSamplerProxy>
164      <hashTree>
165       <HeaderManager guiclass="HeaderPanel" testclass="HeaderManager" testname="AppSensor Header"
166       enabled="true">
167        <collectionProp name="HeaderManager.headers">
168         <elementProp name="" elementType="Header">
169          <stringProp name="Header.name">X-Appsensor-Client-Application-Name2</stringProp>
170          <stringProp name="Header.value">myclientapp</stringProp>
171         </elementProp>
172        </collectionProp>
173       </HeaderManager>
174       <hashTree/>
175       <ResultCollector guiclass="ViewResultsFullVisualizer" testclass="ResultCollector"
176       testname="View Results Tree"
177       enabled="true">
178        <boolProp name="ResultCollector.error_logging">false</boolProp>
179        <objProp>
180         <name>saveConfig</name>
181         <value class="SampleSaveConfiguration">
182          <time>true</time>
183          <latency>true</latency>
184          <timestamp>true</timestamp>
185          <success>true</success>
186          <label>true</label>
187          <code>true</code>
188          <message>true</message>
189          <threadName>true</threadName>
190          <dataType>true</dataType>
191          <encoding>false</encoding>
192          <assertions>true</assertions>
193          <subresults>true</subresults>
194          <responseData>false</responseData>
195          <samplerData>false</samplerData>
196          <xml>false</xml>
197          <fieldNames>true</fieldNames>
198          <responseHeaders>false</responseHeaders>
199          <requestHeaders>false</requestHeaders>
200          <responseDataOnError>false</responseDataOnError>
201          <saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMessage>
202          <assertionsResultsToSave>0</assertionsResultsToSave>
203          <bytes>true</bytes>
204          <sentBytes>true</sentBytes>
205          <threadCounts>true</threadCounts>
206          <idleTime>true</idleTime>
207          <connectTime>true</connectTime>
208         </value>
209        </objProp>
210        <stringProp name="filename"></stringProp>
211       </ResultCollector>
212       <hashTree/>
213      </hashTree>
```

```
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
testname="Report Attack Count" enabled="true">
 <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel"
 testclass="Arguments" testname="User Defined Variables" enabled="true">
  <collectionProp name="Arguments.arguments"/>
 </elementProp>
 <stringProp name="HTTPSampler.domain">localhost</stringProp>
 <stringProp name="HTTPSampler.port">8085</stringProp>
 <stringProp name="HTTPSampler.connect_timeout"></stringProp>
 <stringProp name="HTTPSampler.response_timeout"></stringProp>
 <stringProp name="HTTPSampler.protocol"></stringProp>
 <stringProp name="HTTPSampler.contentEncoding"></stringProp>
 <stringProp name="HTTPSampler.path">/api/v1.0/reports/attacks/count</stringProp>
 <stringProp name="HTTPSampler.method">GET</stringProp>
 <boolProp name="HTTPSampler.follow_redirects">false</boolProp>
 <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
 <boolProp name="HTTPSampler.use_keepalive">false</boolProp>
 <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
 <stringProp name="HTTPSampler.implementation">Java</stringProp>
 <boolProp name="HTTPSampler.monitor">false</boolProp>
 <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree>
 <HeaderManager guiclass="HeaderPanel" testclass="HeaderManager" testname="AppSensor Header"
 enabled="true">
  <collectionProp name="HeaderManager.headers">
   <elementProp name="" elementType="Header">
    <stringProp name="Header.name">X-Appsensor-Client-Application-Name2</stringProp>
    <stringProp name="Header.value">myclientapp</stringProp>
   </elementProp>
  </collectionProp>
 </HeaderManager>
 <hashTree/>
 <ResultCollector guiclass="ViewResultsFullVisualizer" testclass="ResultCollector"
 testname="View Results Tree" enabled="true">
  <boolProp name="ResultCollector.error_logging">false</boolProp>
  <objProp>
   <name>saveConfig</name>
   <value class="SampleSaveConfiguration">
    <time>true</time>
    <latency>true</latency>
    <timestamp>true</timestamp>
    <success>true</success>
    <label>true</label>
    <code>true</code>
    <message>true</message>
    <threadName>true</threadName>
    <dataType>true</dataType>
    <encoding>false</encoding>
    <assertions>true</assertions>
    <subresults>true</subresults>
    <responseData>false</responseData>
    <samplerData>false</samplerData>
    <xml>false</xml>
    <fieldNames>true</fieldNames>
```

```
          <responseHeaders>false</responseHeaders>
          <requestHeaders>false</requestHeaders>
          <responseDataOnError>false</responseDataOnError>
          <saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMessage>
          <assertionsResultsToSave>0</assertionsResultsToSave>
          <bytes>true</bytes>
          <sentBytes>true</sentBytes>
          <threadCounts>true</threadCounts>
          <idleTime>true</idleTime>
          <connectTime>true</connectTime>
        </value>
      </objProp>
      <stringProp name="filename"></stringProp>
     </ResultCollector>
     <hashTree/>
    </hashTree>
    <ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer"
    enabled="true">
     <stringProp name="ConstantTimer.delay">${delay}</stringProp>
    </ConstantTimer>
    <hashTree/>
    <ResultCollector guiclass="RespTimeGraphVisualizer" testclass="ResultCollector"
    testname="Response Time Graph" enabled="true">
     <boolProp name="ResultCollector.error_logging">false</boolProp>
     <objProp>
      <name>saveConfig</name>
      <value class="SampleSaveConfiguration">
       <time>false</time>
       <latency>true</latency>
       <timestamp>true</timestamp>
       <success>false</success>
       <label>true</label>
       <code>false</code>
       <message>false</message>
       <threadName>false</threadName>
       <dataType>false</dataType>
       <encoding>false</encoding>
       <assertions>false</assertions>
       <subresults>false</subresults>
       <responseData>false</responseData>
       <samplerData>false</samplerData>
       <xml>false</xml>
       <fieldNames>true</fieldNames>
       <responseHeaders>false</responseHeaders>
       <requestHeaders>false</requestHeaders>
       <responseDataOnError>false</responseDataOnError>
       <saveAssertionResultsFailureMessage>false</saveAssertionResultsFailureMessage>
       <assertionsResultsToSave>0</assertionsResultsToSave>
       <idleTime>true</idleTime>
       <connectTime>true</connectTime>
      </value>
     </objProp>
     <stringProp name="filename">/home/david/response_time.csv</stringProp>
     <stringProp name="RespTimeGraph.graphsizeheight">500</stringProp>
     <stringProp name="RespTimeGraph.yaxisscalemaxvalue">50</stringProp>
```

```
      <stringProp name="RespTimeGraph.interval">100</stringProp>
      <stringProp name="RespTimeGraph.xaxistimeformat">mm:ss.SSS</stringProp>
     </ResultCollector>
     <hashTree/>
     <CSVDataSet guiclass="TestBeanGUI" testclass="CSVDataSet" testname="CSV Data Set Config" enabled="true">
      <stringProp name="delimiter">,</stringProp>
      <stringProp name="fileEncoding"></stringProp>
      <stringProp name="filename">/home/david/apache-jmeter-3.1/bin/delay.csv</stringProp>
      <boolProp name="quotedData">false</boolProp>
      <boolProp name="recycle">true</boolProp>
      <stringProp name="shareMode">shareMode.all</stringProp>
      <boolProp name="stopThread">false</boolProp>
      <stringProp name="variableNames"></stringProp>
     </CSVDataSet>
     <hashTree/>
    </hashTree>
   </hashTree>
  </hashTree>
</jmeterTestPlan>
```

Appendix E

RELEVANT ALGORITHM METHODS

```
1  public void analyze(Event triggerEvent) {
2    Collection<Rule> rules = appSensorServer.getConfiguration().findRules(triggerEvent);
3
4    for (Rule rule : rules) {
5      if (checkRule(triggerEvent, rule)) {
6        generateAttack(triggerEvent, rule);
7      }
8    }
9  }
10
11 protected boolean checkRule(Event triggerEvent, Rule rule) {
12   Queue<Notification> notifications = getNotifications(triggerEvent, rule);
13   Queue<Notification> windowedNotifications = new LinkedList<Notification>();
14   Iterator<Expression> expressions = rule.getExpressions().iterator();
15   Expression currentExpression = expressions.next();
16   Notification tail = null;
17
18   while (!notifications.isEmpty()) {
19     tail = notifications.poll();
20     windowedNotifications.add(tail);
21     trim(windowedNotifications, tail.getEndTime().minus(currentExpression.getWindow().toMillis()));
22
23     if (checkExpression(currentExpression, windowedNotifications)) {
24       if (expressions.hasNext()) {
25         currentExpression = expressions.next();
26         windowedNotifications = new LinkedList<Notification>();
27         trim(notifications, tail.getEndTime());
28       }
29       else {
30         return true;
31       }
32     }
33   }
34
35   return false;
36 }
37
38 protected boolean checkExpression(Expression expression, Queue<Notification> notifications) {
39 for (Clause clause : expression.getClauses()) {
40 if (checkClause(clause, notifications)) {
41 return true;
42 }
43 }
44
45 return false;
46 }
47
48 protected boolean checkClause(Clause clause, Queue<Notification> notifications) {
```

```
49    Collection<DetectionPoint> windowDetectionPoints = new HashSet<DetectionPoint>();
50
51    for (Notification notification : notifications) {
52      windowDetectionPoints.add(notification.getMonitorPoint());
53    }
54
55    for (DetectionPoint detectionPoint : clause.getMonitorPoints()) {
56      if (!windowDetectionPoints.contains(detectionPoint)) {
57        return false;
58      }
59    }
60
61    return true;
62  }
63
64  protected LinkedList<Notification> getNotifications(Event triggerEvent, Rule rule) {
65    LinkedList<Notification> notificationQueue = new LinkedList<Notification>();
66    Collection<Event> events = getApplicableEvents(triggerEvent, rule);
67    Collection<DetectionPoint> detectionPoints = rule.getAllDetectionPoints();
68
69    for (DetectionPoint detectionPoint : detectionPoints) {
70      Queue<Event> eventQueue = new LinkedList<Event>();
71
72      for (Event event : events) {
73        if (event.getDetectionPoint().typeAndThresholdMatches(detectionPoint)) {
74          eventQueue.add(event);
75
76          if (isThresholdViolated(eventQueue, event, detectionPoint.getThreshold())) {
77            int queueDuration = (int)getQueueInterval(eventQueue, event).toMillis();
78            DateTime start = DateUtils.fromString(eventQueue.peek().getTimestamp());
79
80            Notification notification = new Notification(queueDuration, ``milliseconds'', start, detectionPoint);
81            notificationQueue.add(notification);
82          }
83
84          if (eventQueue.size() >= detectionPoint.getThreshold().getCount()) {
85            eventQueue.poll();
86          }
87        }
88      }
89    }
90
91    Collections.sort(notificationQueue, Notification.getEndTimeAscendingComparator());
92
93    return notificationQueue;
94  }
95
96  protected ArrayList<Event> getApplicableEvents(Event triggerEvent, Rule rule) {
97    ArrayList<Event> events = new ArrayList<Event>();
98
99    DateTime ruleStartTime = DateUtils.fromString(triggerEvent.getTimestamp()).minus(rule.getWindow().toMillis());
100   DateTime lastAttackTime = findMostRecentAttackTime(triggerEvent, rule);
101   DateTime earliest = ruleStartTime.isAfter(lastAttackTime) ? ruleStartTime : lastAttackTime;
102
103   SearchCriteria criteria = new SearchCriteria().
```

```
104       setUser(triggerEvent.getUser()).
105       setEarliest(earliest.plus(1).toString()).
106       setRule(rule).
107     setDetectionSystemIds(appSensorServer.getConfiguration().getRelatedDetectionSystems(triggerEvent.
              getDetectionSystem()));
108
109     events = (ArrayList<Event>)appSensorServer.getEventStore().findEvents(criteria);
110
111     Collections.sort(events, Event.getTimeAscendingComparator());
112
113     return events;
114
115   }
```