3-23-2017

Active Response Using Host-Based Intrusion Detection System and Software-Defined Networking

Jonathon S. Goodgion

ACTIVE RESPONSE USING HOST-BASED INTRUSION DETECTION SYSTEM AND SOFTWARE-DEFINED NETWORKING

THESIS

Jonathan S. Goodgion, 2d Lt, USAF

AFIT-ENG-MS-17-M-032

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-17-M-032

ACTIVE RESPONSE USING HOST-BASED INTRUSION DETECTION SYSTEM AND SOFTWARE-DEFINED NETWORKING

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Jonathan S. Goodgion, B.S.C.S.

2d Lt, USAF

March 2017

AFIT-ENG-MS-17-M-032

ACTIVE RESPONSE USING HOST-BASED INTRUSION DETECTION SYSTEM AND SOFTWARE-DEFINED NETWORKING

THESIS

Jonathan S. Goodgion, B.S.C.S.
2d Lt, USAF

Committee Membership:

Barry E. Mullins, Ph.D., P.E.
Chair

Timothy H. Lacey, Ph.D., CISSP
Member

Michael R. Grimaila, Ph.D., CISM, CISSP
Member

AFIT-ENG-MS-17-M-032

Abstract

Recent studies have shown information misuse is more abundant *within* organizations, possibly from the modern proliferation of phishing and social engineering attacks. To address this problem, security experts have proposed local, internal inspection, known as Host-based Intrusion Detection Systems (HIDS). This technique allows detection of malicious activity to occur at the endpoints – the individual hosts.

This research proposes Active Host-based Network Security Response (AHNSR): a framework that utilizes HIDS with Software-Defined Networking (SDN) to enhance system security by allowing dynamic active response and reconstruction from a global network topology perspective. Vital to the operational architecture, SDN introduces new programming capabilities that resolve the rigid configurations of traditional networks. This flexibility is essential in an era of rapidly evolving threats that require unique and adaptive security defenses, as SDN-software solutions enable more flexible security controls in both virtualized and enterprise environments. These SDN security controls aim at providing access to legitimate users, protecting systems from attacks, and providing mitigation or countermeasures when attacks do occur.

AHNSR is a complete and automated dynamic security solution, starting with alert generation and ending with an appropriate SDN-enabled response on physical switch flow tables. Responses include traffic redirection, host quarantining, filtering, and more. SDN allows greater flexibility and command/control compared to legacy, manual administrator response or the iptables solution of traditional IDS. By pushing security event information to a global controller, immediate and intelligent changes can be made for packet forwarding devices across the network. When, not *if*, an attack occurs, AHNSR increases responsiveness while decreasing information damage

– a goal every incident response team should have.

This research compares different methods of pushing security event information from hosts and the resulting system performance at specified security level thresholds. A testable SDN-controlled network is constructed with multiple hosts, OpenFlow enabled switches, and a Floodlight controller, all linked to a custom, novel interface for the Open-Source SECurity (OSSEC) HIDS framework. OSSEC is implemented in a server-agent architecture, allowing scalability and OS independence. System effectiveness is evaluated against the following factors: alert density and selective Floodlight module response types.. At the expected operational load of 500 events per second (EPS), results reveal a mean system response time of 0.5564 seconds from log generation to flow table update via Floodlight's Access Control List module. These results demonstrate the AHNSR framework as a dependable (100% successful alert generation), effective (under 1 second), and efficient (under 35% CPU utilization) security solution in the emulated test network. Load testing further assesses performance of response methods at EPS levels 10—10000. A dynamic model is fit against the data, allowing for performance estimates of dynamic response policies found in real enterprise environments.

Acknowledgements

I am continually humbled by the blessings I receive, and must first give thanks to the Lord for providing it all.

I have learned so much in this program thanks to the educational passion demonstrated by AFIT faculty members like Dr. Barry Mullins, Dr. Timothy Lacey, and Dr. Michael Grimaila. They help make AFIT an institution that truly fosters, challenges, and mentors the future leaders of this great nation.

Lastly, I am beyond grateful for my beloved wife and her unending support. Thank you for being a light in my life, helping discern the Lord's will for us in the marvelous mission to share love everywhere.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACTIVE RESPONSE USING HOST-BASED INTRUSION DETECTION SYSTEM AND SOFTWARE-DEFINED NETWORKING

I. Introduction

1.1 Background

The world is rapidly emerging as a network of networks, with millions of packet forwarding devices responsible for the successful delivery of any request across the Internet. However, traditional IP network configuration is complex, and once properly configured, usually remains as a semi-static architecture. Inevitably, the exponentially-advancing technology in the current age quickly outgrows the static limitations of traditional networks. Software-Defined Networking (SDN) is a possible solution to this problem. It is an emerging networking paradigm that gives hope in solving the limitations of current network infrastructures.

1.2 Problem Statement

According to annual cyber trend reports, half a billion personal records were stolen or lost in 2015 [8]. There is no argument here: any "hackivist" or state-sponsored entity has free reign on poorly-configured and vulnerable systems. Even well-vetted commercial cyber defense systems can fall victim to a Zero-Day vulnerability or Advanced Persistent Threat (APT). Additionally, there is a trend of increasing insider involvement in data breaches; while it only accounted for around 10 percent of data breaches in 2014, insider involvement was present in 32 percent of the claims submitted in 2015. With more than three-quarters of surveyed US government agencies admitting they are more focused on combating insider threats today than one year ago, it corroborates the significance of the problem.

Even though threat detection is possible through various Commercial-Off-The-Shelf (COTS) products, it remains ineffective. Unfortunately, law enforcement and third party notification remained the top breach discovery methods in 2015 [8]. In 2015, 92.9% of compromises took minutes or less to complete, and when companies have to wait on external notification, it's too late to secure, respond, and mitigate the active threat [9]. Therefore, it is apparent there is a problem in real-time threat detection and response from both external and internal originating intrusions.

Software-Defined Networking introduces a new framework for network configuration, security, and management. Researchers are investigating and developing a wide array of innovative and effective tools that utilize the fundamental architecture of a global controller in a Software-Defined Network. These new tools may help prevent and effectively respond to intrusions. When attackers gain unauthorized access to any given machine, their most common goal is to move laterally to extract the most information possible. Thus, the most effective methods in reducing data loss and damage are 1) preventing lateral access to locally networked machines and 2) responding to the breach as soon as possible.

1.3 Research Goals

In response to the problems of threat detection and response, this research seeks to expand on a process of actively responding to local host-based security violations by utilizing key technologies provided by Host-based Intrusion Detection Systems in combination with Software-Defined Networking. The ideal system would integrate seamlessly with existing systems, provide administrators with greater network control and insight, and significantly reduce the possibility of data loss/theft while maintaining access to critical network services.

The research assesses the performance of four different methods to SDN response demanding alerts, in which immediate network reconfiguration by a SDN controller helps mitigate a potential active threat or intrusion. An open-source Host-based Intrusion Detection System provides the detection and alert mechanism, whereas the SDN controller is responsible for executing the response with flow table updates. The SDN controller Application Programming Interfaces (APIs) provide the four evaluated methods of responding to an alert: Firewall, Access Control List (ACL), Static Flow Entry, and Log-Only. Each method helps meet specific data security compliance requirements, such as those required by Payment Card Industry Data Security Standards (PCI DSS) 2/3.2 and the Health Insurance Portability and Accountability Act (HIPAA), which are furthered discussed in Section 2.7 [10] [11].

Specifically, this research seeks to expand on Todd's Dynamic Security Control System (DSCS), discussed in Section 2.9 [6]. The limitations of Todd's previous work in this area have been quantified, and this research seeks to mitigate shortcomings and expand functionality in the following ways:

Limitation Mitigations

- The use of Floodlight's stateless firewall could not interrupt active connections if they were found to be malicious. This research seeks a viable solution.
- The DSCS system was not interfaced with an actual IDS. This research seeks to use industry standard software (OSSEC) and its APIs to generate alerts.

Improvements

- A database schema is incorporated to allow alert sharing and archival tracking, meeting standard logging policy requirements and improving scalability.

- Different approaches to intrusion response (firewall or access control list updates, first hop modifications, honeypot redirection, etc.) are allowed for more dynamic control.
- The AMQP publisher/broker model can be replaced with a similar messaging design within a Security Incident Management (SIM) or Security Information and Event Management (SIEM) product.
- The programmability of the AMQP protocol can be replicated in a SDN environment because routing decisions are defined outside the network layer of packet forwarding devices.

1.4 Hypothesis

This research hypothesizes as the aggregate number of logs being generated by hosts increases, the response time will increase, the processor resource utilization will also increase, and the alert generation rate will remain above the protocol standard of 98%. At a normal operational load level, this research expects the average event response time to be under 1 second for near real-time reactivity. Additionally, when comparing the different response methods, the Access Control List response method will outperform the Firewall response method in both quantitative and qualitative metrics.

1.5 Approach

An experimental fully-switched network is configured, consisting of ten virtual agents with host-based intrusion monitoring, one management server, and one SDN controller. In order to emulate a full SDN network, packet forwarding is restricted to OpenFlow enabled switches on real hardware. High resolution timing scripts are deployed and log generation is controlled while monitoring system performance through the detection/response workflow. An active response script is developed to query different modules provided by the SDN controller.

1.6 Assumptions/Limitations

The following assumptions/limitations are understood when designing and executing experiments for the active response system:

- The framework is dependent on well-configured host auditing and logging policies, as well as appropriate rule definitions on the management server.
- Triggers and definitions are limited by the features provided by the Host-based IDS utilized in the research, Open Source SECurity (OSSEC), version 2.9 RC3. More specifically, alert triggers are limited to system log analysis, application log analysis, connection status, rootkit signature detection, selective registry changes, and/or selective file changes.
- Rule definitions are limited to the following: alert level, specific rule ID, event frequency, maximum size, time frame, category, source IP, destination IP, username, hostname, program name, URL, and/or matching regex parameters.
- This research does not examine identification accuracy (e.g., false positive frequency) or use any datasets of captured attacks.
- The Floodlight controller has out-of-band management to all switches in the network. If this is not true, it does not ensure dedicated OpenFlow channels that may become congested otherwise, which can invalidate the observed results.
- All switches must support OpenFlow 1.3 or higher, as some response modules require features in these versions.

- A pre-configured flow is utilized for proactive forwarding to the IDS server. This would not be possible in a pure reactive forwarding environment.
- CPU utilization data collection frequency is limited to at most once per second. This reduces the sample size for this metric significantly and can potentially be inaccurate or inconsistent between trials.

1.7 Contributions

This thesis contributes to the general SDN body of research. It helps answer questions about how SDN can be utilized in further securing local area networks by providing interfaces to already existing systems. It provides empirical evidence that a complete SDN-interfaced intrusion detection *and* response action is capable of actively mitigating threats in near real-time.

1.8 Thesis Overview

This thesis document is arranged in six chapters. Chapter 2 presents fundamental concepts of networking, SDN, IDS, and other relevant research. Chapter 3 discusses the system design details and the necessary configuration in implementing an active response. The experiment methodology is presented in Chapter 4 along with the standard parameters, metrics, and testing process. Chapter 5 covers the results and necessary statistical analysis of the collected data. Lastly, Chapter 6 summarizes the research and discusses potential opportunities for future work in this domain.

II. Background and Related Research

2.1 Overview

This chapter discusses Software-Defined Networking (SDN), Intrusion Detection Systems (IDS), and the relevant bodies of research. Section 2.1 discusses the history of networking and the motivation of SDN development, while Section 2.2 further explains SDN fundamental components. Several application and use case proposals are addressed in Section 2.3, along with current research trends in this domain. One of these areas is security, which is discussed further Section 2.4, from both potential weakness and strength perspectives. In Sections 2.5 and 2.6, high-level concepts of Host-based Intrusion Detection Systems and the Advanced Message Queuing Protocol are introduced respectively, as they play key roles in the research methodology for security policy enforcement using SDN.

2.2 Networking

In order to reliably send data from one host to another, computers depend on switches, routers, and other packet forwarding devices to pass along the message. This is necessary for both inter *and* intra network communication. Data is encapsulated into packets and sent across various mediums utilizing transport and network layer protocols, most notably the Transport Control Protocol (TCP)/Internet Protocol (IP). These protocols establish a standard "language" and delivery method for the majority of traffic on the Internet. However, all machines cannot be directly connected to one another. Thus, packet forwarding devices act as the highway intersections, directing traffic towards individual, appropriate destinations.

Computer networks can be divided into three functional planes: data, control, and management [12]. The data plane is where the networking devices reside and is where the raw bits are forwarded through to the next device. The control plane represents the protocols used to populate the forwarding tables of the data plane elements. Lastly, the management plane represents the services, software, and protocols used to remotely monitor and configure the control plane. Figure 1 depicts the combination of these three planes in a top-down, streamlined process of policy development to policy execution. Network policy is defined in the management plane, which is enforced via the control plane and ultimately executed at the physical data plane.

The most significant limitation of current packet forwarding devices is caused by their implementation of these three functional planes. Switches were designed to be resilient and independently capable, which resulted in a tight coupling of the data and control planes. Therefore, the routers are responsible for both packet forwarding (data plane), as well as managing a variety of distributed routing algorithms (control plane). The control plane is responsible for *any* extra processing (e.g., Link Layer Discover Protocol, Border Gateway Protocol) that is necessary to perform the basic data plane activities. Unfortunately routers are doing all of this extra work themselves in tracking topology changes, computing routes, and installing forwarding rules. In this framework, the devices themselves are independently responsible for both managing and executing their own forwarding tables. This breeds a decentralized, static

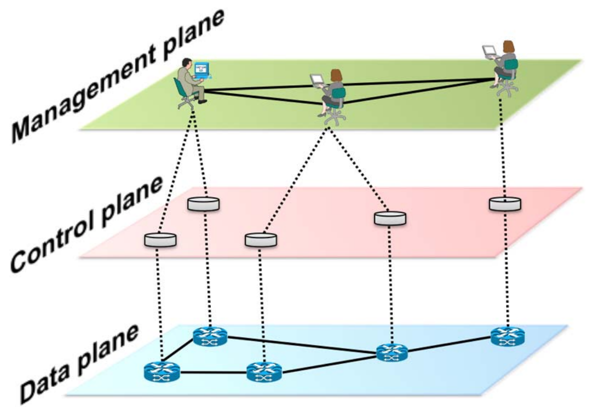


Figure 1. Layered view of the functional networking planes [1]

architecture – a contributing factor to the difficulty of innovation in this domain.

2.2.1 History.

There have been three distinct phases in the demands of networking over the past 20 years. Beginning in the late 1990s, as the Internet became a consumable service in both business and home, the focus was almost exclusively on bandwidth. Data rates were the only metric in considering the performance of file transportation from point A to point B. It did not matter *how* data got to a destination. However, as more services and protocols were introduced in networking, specifically Real-Time Communication (RTC) traffic, a new performance metric was introduced: latency.

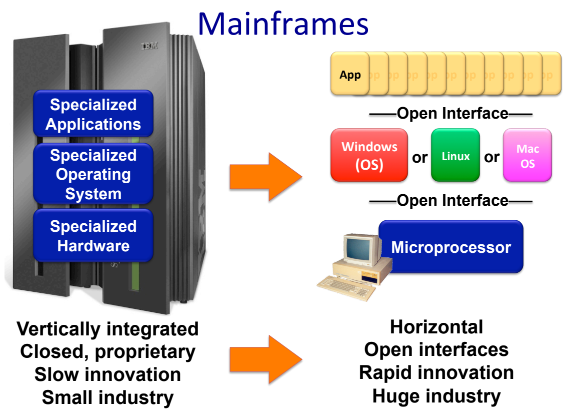
In the 2005 time-frame, with the advent of video and audio streaming services such as YouTube, Skype, and Voice Over IP (VoIP), network administrators began to realize that latency was a real challenge with these services. This problem occurs because the service depends on delivery *timing* of individual packets, far different from the former concept of getting an entire file to a destination regardless of actual semantics. But as internal network RTC traffic volume increased, competition of those "intersection" resources at packet forwarding devices demanded the need to prioritize traffic type. As a result, Quality of Service (QoS) was introduced to prioritize the forwarding of individual packets (e.g., RTC traffic) to reduce latency for those services.

Moving forward to present-day, networks have become increasingly complex. With more devices and more services than ever before, it is becoming clear that QoS standards cannot accurately prioritize traffic all of the time. It is limited by strict assumptions, and fails to allow administrators to dynamically model and shape their traffic depending on the real-time needs of a business. There is a desperate need to programmatically respond to network resource demand by modifying certain prioritization and flow controls.

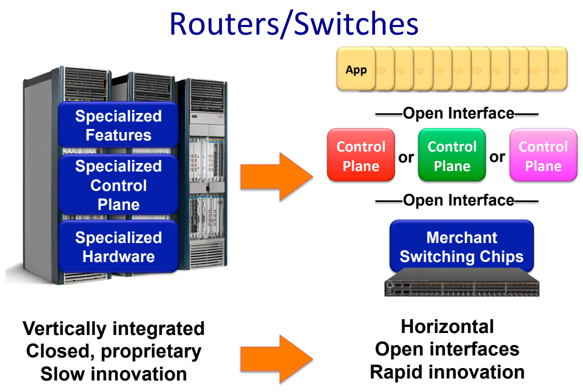
2.2.2 SDN Origination.

SDN, in its simplest form, is pulling away the intelligence from switch hardware. This trend is already occurring in other domains with positive results. For example, file servers were originally dedicated to a particular service and all permissions, data, assets, and physical components were managed in a self-contained unit. These servers have been replaced by Storage Area Networks (SAN), allowing a consolidation of file systems to abstract the connections from a single operating system [13]. Similarly, virtualization tools are helping create multiple instances of software operate on the same hardware.

Investigating the history of basic computing reveals this trend. Nick McKeown highlights in Figure 2 the similarities between legacy mainframe computers and traditional networking devices [2]. Both incorporate specialized applications, operating systems, and hardware in a vertically-locked environment. The shift to personal computing was revolutionary, yet required the separation of hardware and intelligence. While this trend grew rapidly in these other domains, the traditional IP networking architecture failed to follow suit – until SDN was introduced.



(a) Mainframe computer framework



(b) Traditional networking framework

Figure 2. Decoupling applications, operating systems, and hardware in network devices is similar to the transition in personal computing [2]

SDN originated from the work at Stanford University in 2007 by researcher Martin Casado [14]. His team successfully decoupled the control and data planes, resulting in the development of OpenFlow (OF), an open-source communication protocol used in the "southbound" control channel, which has become the standard interface between servers and SDN-enabled switches. Shifting the intelligence away from the physical forwarding devices allows more flexibility in networking configuration than ever before.

According to the Open Networking Foundation (ONF), SDN is defined as a network architecture that "decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services" [15]. Kreutz *et al* go further in [1], defining SDN with four pillars:

1. The control plane functionality is decoupled from the data plan network devices.
2. Forwarding decisions are "flow" based versus strictly destination based (flows are defined in Section 2.3.2).
3. Control logic is executed on an external device: the SDN controller.
4. Software applications running on top of the SDN controller are capable of interacting with the underlying data plane devices.

When a network builds upon these four pillars, it successfully forces the separation of management and execution.

2.3 SDN Components

By nature of the four pillars described, SDN requires distinct components that have already been mentioned in theory. Figure 3 depicts the SDN architecture and its fundamental components. This section discusses the three central components unique to SDN: the Network Operating System, the southbound interface, and the northbound interfaces.

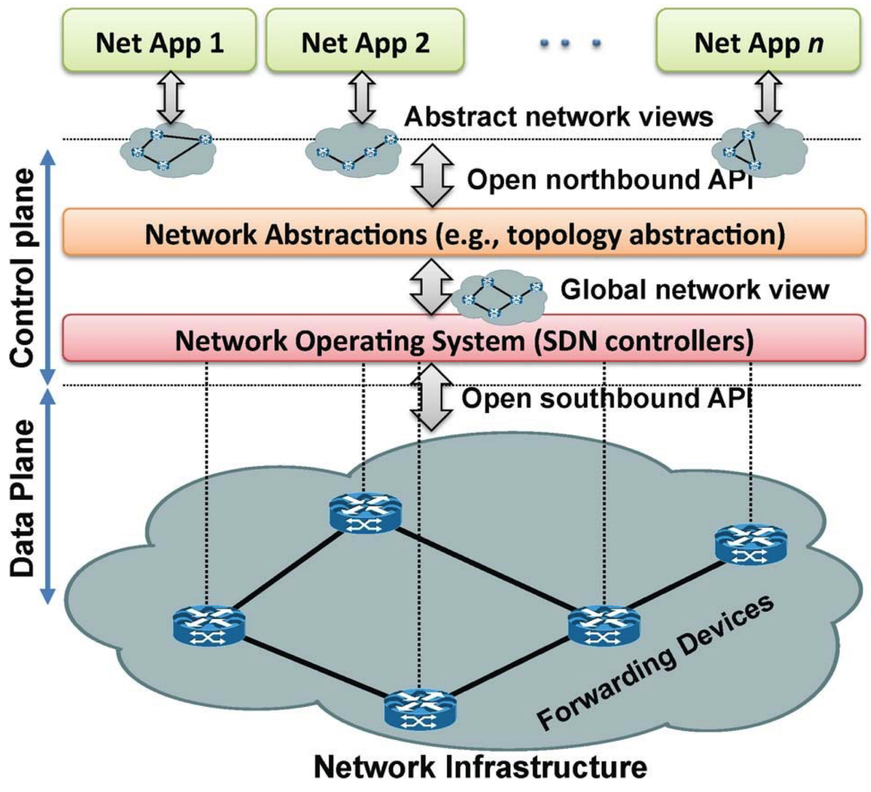


Figure 3. SDN components [1]

2.3.1 Controller: A Networking Operating System.

The controller, also known as the Network Operating System (NOS), is arguably the most important component in the SDN architecture. The purpose of any operating system is to provide abstraction to higher level applications and communication to lower level components. It is also responsible for managing those resources and services. SDN incorporates this same paradigm in delivering and facilitating network management. At a minimum, a NOS provides the following functionality: maintain network state and topology information, discover new devices, and distribute network configuration settings [12].

When designing the NOS architecture, the most significant consideration is whether it operates in a distributed or centralized structure. A centralized controller manages all data plane devices as a single entity. NOX, Floodlight, and Beacon have been designed as such [15]. However, centralized systems may introduce danger as a single point of failure; this threat has motivated others to develop distributed NOS controllers. Another advantage of a distributed system is scalability. Onix, ONOS, and Hyperflow are examples of distributed controllers, all capable of higher degrees of availability and resiliency by their fault tolerant design [16].

In this research, the Floodlight controller is used. It is an enterprise-class, Apache-licensed, and Java-based OpenFlow controller. This controller is selected due to its community support, open-source code base, centralized design, and implementation of a northbound interface.

2.3.2 Southbound Interface.

The southbound interface is the link between the NOS and the packet forwarding devices. The most commonly accepted protocol for this communication is Casado's OpenFlow, which provides three sources of information for NOSs [14]. First, reactive messages are sent to the controller whenever links drop or port changes occur. Second, flow statistics and corresponding updates are sent to the controller. Last, "packet-in" messages are sent across the southbound interface whenever the switches/routers do not know what action to take on an incoming flow.

A flow is an important concept in SDN, and consists of three parts: 1) a matching rule, 2) actions to perform on matching packets, and 3) counters for statistical use. Flow generation is accomplished by utilizing the Ternary Content Addressable Memories (TCAMs) that are already implemented by forwarding tables in typical modern switching equipment [17]. The advantage in SDN is that flows are generated with a global perspective of the network. Flow rules are determined at the control and management level, and then disseminated throughout the network.

Figure 4 shows how upon packet arrival, if a match is determined against a rule in a flow entry, the corresponding action occurs. For example, it can forward to either a physical or virtual port, en-queue to a particular port, encapsulate and send to the controller, modify existing fields, or simply drop the packet. "Table Chaining" also introduces hierarchical look-up options by having an action link to another flow table or even a group table, defined as a specialized table which resembles a set of multiple actions to execute [1].

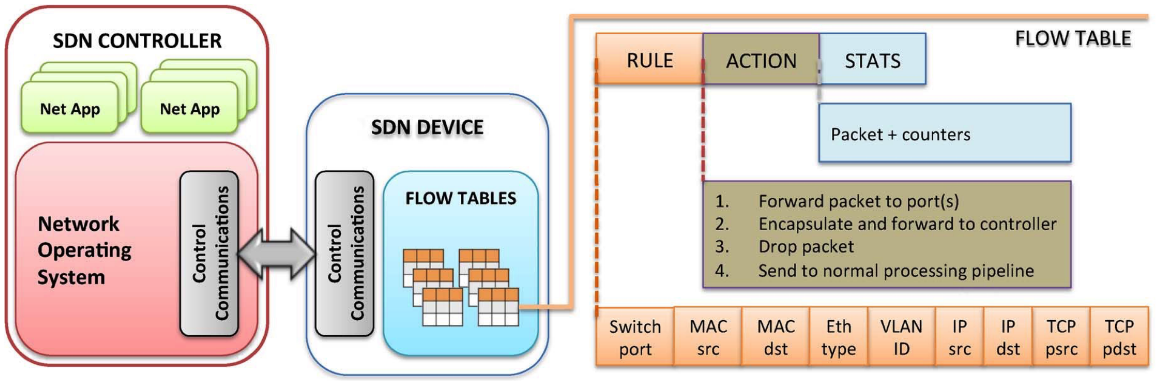


Figure 4. How flow tables are used in the SDN architecture [1]

Flow entries may be issued on a hard timeout or idle timeout basis. A hard timeout signals the switch or router to remove the entry after its single use, while an idle timeout allows the entry to remain until it has had no matches for a certain amount of time. Timeouts are helpful because a single flow can be decided once against the first packet in a targeted TCP connection and then used repeatedly for subsequent forwarding, increasing performance.

2.3.3 Northbound Interface.

The northbound interface is mainly a software domain that allows network applications to function independent from the lower level SDN implementation, but also enables dynamic control of actual network resources. The Open Networking Foundation (ONF) framework describes customer and business applications having abstracted views and the ability to program specific needs across multiple domains [15].

While OpenFlow has emerged as the *de facto* standard on the southbound interface, there is not an equivalent Application Programmable Interface (API) adoption for its counterpart north side. Whatever the case may be, an open-source northbound interface is extremely important in keeping true portability and interoperability among SDN platforms.

2.3.4 Special/Optional Components.

Depending on the SDN deployment environment, there may be special requirements or optional features that help improve performance. For example, in a distributed controller environment, East/Westbound APIs are a special component involved. This horizontal interface is responsible for exchanging data between controllers, running consistency checks, and monitoring controller function status. SDNi is a protocol developed to coordinate flow setup across these distributed platforms [18].

In addition, the concept of virtualization allows an entire SDN to live as a virtual network. Open vSwitch is an open source virtual switch that can run as a standalone hypervisor switch, or as a distributed switch across multiple servers. The virtual switching occurs within the Virtual Machine Manager (VMM) and can therefore avoid physical hardware limitations. For example, Open vSwitch uses Link Aggregation

Control Protocol (LCAP), a method of combining multiple physical ports to form a single logical channel in parallel [19].

2.4 Use Cases and Applications

The programmability of SDN has presented a variety of use cases for real world scenarios. This section discusses a few examples, and explains how the benefits of SDN contributes to their potential.

2.4.1 Access Control.

In a dynamic access control model, SDN allows forwarding devices to “inspect” the first packet on any connection. They can then consult the access control policy via pre-determined flows in their tables (proactive modeling) or request an access policy from the controller (reactive modeling). SDN allows these rules to be instantiated at a global level to block or route traffic appropriately, and do not require manual configuration at each switch/router in the internal network. This research utilizes Floodlight’s proactive access control module as one response method.

2.4.2 Mobility and Migration.

SDN provides the ability for seamless migration of mobile clients. If a host migrates to a new location on the network, controller topology visibility is able to signal the change, and modify rules/routing information to optimally reroute traffic immediately.

2.4.3 Fault Tolerance.

A major issue with Shortest Path First (SPF) routing algorithms found on many networks today is the problem of non-deterministic failures. The failure of a link between two nodes affects all nodes that use this link to reach some destinations. These nodes will change their routing tables and send out routing updates the first-hop devices, essentially competing over alternate links in the network. Latency is involved during the rebuild, and the algorithm's non-deterministic nature does not guarantee the rebuild is the most optimal solution [20].

Alternatively, SDN guarantees fault tolerance by having an accurate global topology. When a link fails in a SDN environment, the controller can immediately establish optimal new flows for all connections affected by the fault.

2.4.4 Dynamic Service Bandwidth.

Bandwidth throttling is a current network practice designed to place limitations on the data capacity of certain services. For example, a company may allocate a maximum of 15% of their total bandwidth for File Transfer Protocol (FTP) traffic, 25% for VoIP traffic, and so forth. Even large cellular providers like T-Mobile automatically throttle bandwidth demanding video streaming traffic down to 480p resolution quality [21]. These throttles are strict boundaries manually configured at the currently coupled data/control plane. A consequence of bandwidth throttling is the lack of flexibility and utilization efficiency. The system is difficult to change and often results in unused bandwidth.

SDN offers the flexible and efficient solution of dynamic service bandwidth. This can meet the changing demands of a company's network as it grows, adds services, and responds to certain events. Additionally, bandwidth *requesting* becomes possible, where a traffic engineering server (possibly integrated into the NOS) receives bandwidth requests from applications. This reactive model presents a new concept of "on-demand" networking, where the network shapes itself based upon endpoint devices, considering their roles and needs.

Furthermore, Eli Etherton presents a possible use case of this dynamic control as an emergency alert system [22]. At the push of a button, SDN can shift all internal bandwidth to deliver pertinent emergency information, possibly allowing real-time high definition streaming to all end users of emergency notifications and instructions. Current networks would choke under QoS and divisional constraints, but SDN's adaptive abilities can overcome those limitations.

2.4.5 Current Research Areas.

SDN is still in the early stages of adoption and standardization. However, there are some themes in current research areas [1].

- **Scalability**: Partitioning and replicating the controller state to segment management as networks grow [23].
- **Testing and Debugging**: Since the network depends on the execution of a program, bugs can be introduced. There is a need for testing techniques specific for controller applications [24].
- **Network Function Virtualization (NFV)**: Moving current application layer services (DHCP, DNS, etc.) to layer 3 network services [25].
- **Migration**: Implementing hybrid deployment environments to ease the transition into a full-scale SDN replacement [26].
- **Software-Defined Environments (SDE)**: Combining SDN with software-defined storage, software-defined computation, and software-defined management [27].
- **Security and Dependability**: Using technologies to improve resiliency and mitigating unrealized security threats on the control platform [28] [29].

This research looks at the themes of Security, Dependability and NFV by moving network security services closer to the network controller.

2.5 Security

While research and experimentation continue to develop with SDN, security continues to be a top priority. The ability to view network status in real-time and programmatically control network behavior is a two sided coin: it opens new avenues of attack vectors for malicious users, but also grants the possibility of providing further software-based security services. Ali *et al* divide current SDN-based security into two categories: protect the network or provide security as a service [30].

2.5.1 Attack Vectors.

There are three new attack vectors introduced in an SDN environment [31]:

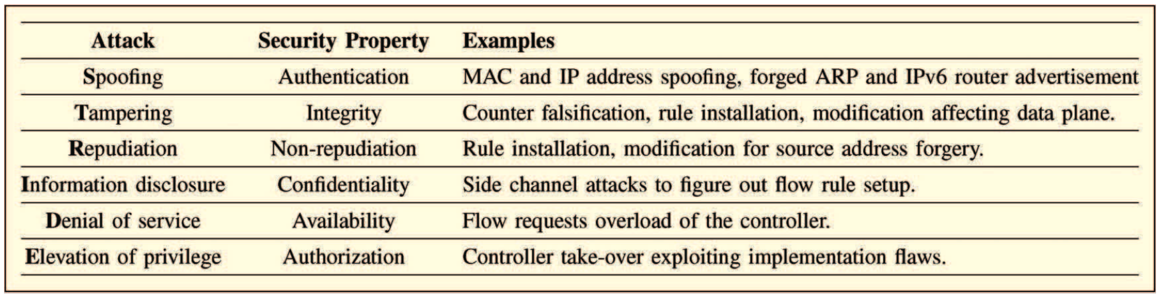
1. **Controller**: As mentioned earlier, pulling the network intelligence to a single system naturally makes the controller a high profile target. If the NOS is compromised, the forwarding devices it controls will be affected as well.

2. **Southbound Interface**: This is an interesting attack vector because it takes advantage of the fact that control messages must now occur between devices and the controller. If this OpenFlow communication is not sent over a secure channel, malicious users may be able to apply traditional techniques such as spoofing and man-in-the-middle (MITM) attacks to issue their own commands to switches and/or routers.

3. **Flow Entries**: Since flow rules are established at the management level, there may be errors in policy and decision making. Therefore, attackers may attempt to reverse engineer the flow tables and pinpoint weaknesses in the actual

implementation.

Additionally, a security assessment of the OpenFlow protocol presented a number of issues. The STRIDE methodology [3], summarized in Figure 5, identifies different attacks, examples, as well as their properties in reference the Confidentiality, Integrity, and Availability (CIA) triad [32].



Attack	Security Property	Examples
Spoofing	Authentication	MAC and IP address spoofing, forged ARP and IPv6 router advertisement
Tampering	Integrity	Counter falsification, rule installation, modification affecting data plane.
Repudiation	Non-repudiation	Rule installation, modification for source address forgery.
Information disclosure	Confidentiality	Side channel attacks to figure out flow rule setup.
Denial of service	Availability	Flow requests overload of the controller.
Elevation of privilege	Authorization	Controller take-over exploiting implementation flaws.

Figure 5. Possible attacks on OpenFlow enabled networks [3]

2.5.2 Securing Using SDN.

Despite the introduction of new attack vectors, SDN also provides the ability to build application-level security for an entire network. The first direction of security mitigation efforts focuses on threat detection, remediation, and verification using SDN.

As examples, the controller may be able to infer a Denial of Service (DoS) attack from the internal network state, and respond by dynamically reprogramming the devices at gateway locations to drop the malicious traffic flow. RadWare has developed DefenseFlow [33], a commercially-available SDN application that does exactly this. DefenseFlow instructs the controller to collect statistics on traffic flow at a per second resolution, establishing baseline measurements to detect any patterns of suggestive DoS attacks.

Dealing with threat remediation, the only possible response in traditional networks is to drop the offending the traffic. SDN, however, with its real-time programming ability, gives a richer set of dynamic responses, including "emergency alarms, dynamic quarantine solutions, traffic redirection for forensics, and entrapment mechanisms such as tarpits and honeypots" [30]. For instance, the FRESCO Application layer prototype, propose by Shin *et al*, provides a Python scripting API that allows programmers to develop reusable flow rule configuration modules. These modules are overseen by FORTNOX, a specialized security enforcement kernel supplement in the controller. Using FRESCO, the authors built Reflector Net, an application that detects and reroutes malicious scanners to isolated honeynets [34]. That traffic can then be dissected, promoting defense intelligence, instead of being immediately dropped at the network perimeter.

SDN can also provide subscription-like security, enabling services such as anonymization, enhanced trust, and remote management. AnonyFlow is a service that theoretically allows an Internet Service Provider (ISP) to dynamically and temporarily assign IP addresses and flow-based identifiers to their customers [35]. This would allow ISPs to mimic hierarchical network address translation (NAT) for entire regions or sub-regions.

SDN can also be utilized to secure data offloading from mobile and Bring-Your-Own-Devices (BYOD). There may exist a need to offload sensitive data from these devices for further processing, sharing, or archival. Enterprise-Centric Offloading System (ECOS) is an enterprise-wide solution, enabling a SDN controller to negotiate with mobile applications on data offloading, encryption feasibility, privacy requirements, and more [36].

2.6 Intrusion Detection Systems

In [37] Kizza describes the fundamentals of network intrusion detection and prevention. An intrusion is defined as a deliberate attempt, successful or not, to break into, access, manipulate, or misuse some valuable property. The Department of Defense (DoD) is responsible for an extensive amount of valuable property, in the form of national intelligence, personnel files, and any other classified information.

To protect this data, one or many Intrusion Detection Systems (IDS) are used to detect unauthorized intrusions into a network. Historically, Network-based Intrusion Detection Systems (NIDS), like SNORT [38], have been used to monitor network traffic from an external, passive perspective. Port mirroring is often utilized to submit network traffic to an IDS, which will then perform an anomaly and/or signature-based analysis.

However, recent studies [39] [40] [41] have shown that information misuse is more abundant *within* organizations, possibly from the modern proliferation of phishing and social engineering attacks. To address this problem, security experts have proposed local, internal inspections, known as Host-based Intrusion Detection Systems (HIDS). This technique allows detection to occur at the endpoints, on individual hosts. This research utilizes an HIDS framework introduced in Section 2.7.

2.6.1 Alert Generation.

HIDS use software to monitor specific logs on the target systems. There are three models of generating alerts:

1. **Anomaly-based detection:** These systems compare network activity to a fingerprint profile of "normal" activity. Any user behavior or system resource usage that deviates from the norm may indicate an intrusion. Training datasets are often used to establish the normal baseline.

2. **Signature-based detection:** These systems assume each intrusive activity can be individually identified by a unique pattern or signature. They work by

keeping an updated database of known malicious signatures and looking for these in current network activity.

3. **Hybrid detection:** Because of the potential shortcomings in both anomaly and signature based detection, a hybrid model combines both sources for alert generation.

The HIDS in this research utilizes a hybrid model in generating alerts.

2.6.2 Incident Response.

While much time is spent optimizing intrusion detection and prevention systems, there is little guidance offered for when an incident does occur. Network administrators have to invest in building a custom incident response strategy; if this is weak, it undermines the entire protection infrastructure.

The SANS institute suggests a simple tiered response and escalation procedure for detected intrusions [42]:

- **Level 1:** "Unfriendly" but harmless activity, such as port scanning. The response should be 1) record user/IP/domain of intruder, and 2) maintain vigilance for future intrusions.

- **Level 2:** An unsuccessful, yet clear attempt to obtain unauthorized information (password files, access restricted hosts, etc.). The response should be 1) research the attack origin, 2) analyze results, and 3) identify intrusion method risks.

- **Level 3:** Serious or successful attempt to breach security. The response should be 1) contain the intrusion immediately, 2) collect and protect information, 3) eliminate related vulnerabilities.

2.7 OSSEC: Open Source SECurity

OSSEC is an open-source HIDS platform capable of incident detection, response, and information management. It functions in either a local or server/agent deployment and can provide host-based monitoring across Linux, Solaris, AIX, HP-UX, BSD, Windows, Mac and VMware ESX systems. OSSEC is used by Internet Service Providers (ISPs), universities, government agencies, and even large corporate data centers (Atomicorp, Wazuh) as their primary HIDS solution [43]. AlienVault's Unified Security Management (USM) subscription product also uses OSSEC in their core functionality.

OSSEC helps customers meet file integrity monitoring, log inspection/monitoring, and policy enforcement requirements by providing these key features [43]:

1. **Log Analysis:** OSSEC collects, analyzes, and correlates logs generated by operating systems, applications, and devices. This feature covers PCI DSS Section 10.

2. **File Integrity Monitoring (FIM):** Independent hosts can detect changes to any file and provide alerts when changes occur. This feature covers PCI DSS Sections 11.5 and 10.5.5.

3. **Registry Integrity Checking:** Similar to file integrity monitoring, agents can run checks against system registry values and alert when changes occur. This feature also covers PCI DSS Sections 11.5 and 10.5.5.

4. **Host-based Anomaly Checking:** OSSEC runs UNIX-based rootkit detection scripts that can detect when a system is modified in a way matching common rootkit methods. For example, it looks for the presence of hidden processes by using *getsid()* and *kill()* to check if any pid is being used or not. If the pid

is being used, but “ps” cannot see it, this likely indicates the presence of a kernel-level rootkit or a trojaned version of “ps”.

5. **Active Response:** This feature allows OSSEC to take immediate action when any specific (or generic) alert occurs.

Figure 6 shows OSSEC’s structure and delegation between the management server and agents. Agents collect log files, command output, and maintain a small database for file integrity checking. However, the server is responsible for analysis through log decoding, archiving alerts, alerting through email, and initiating active response scripts.

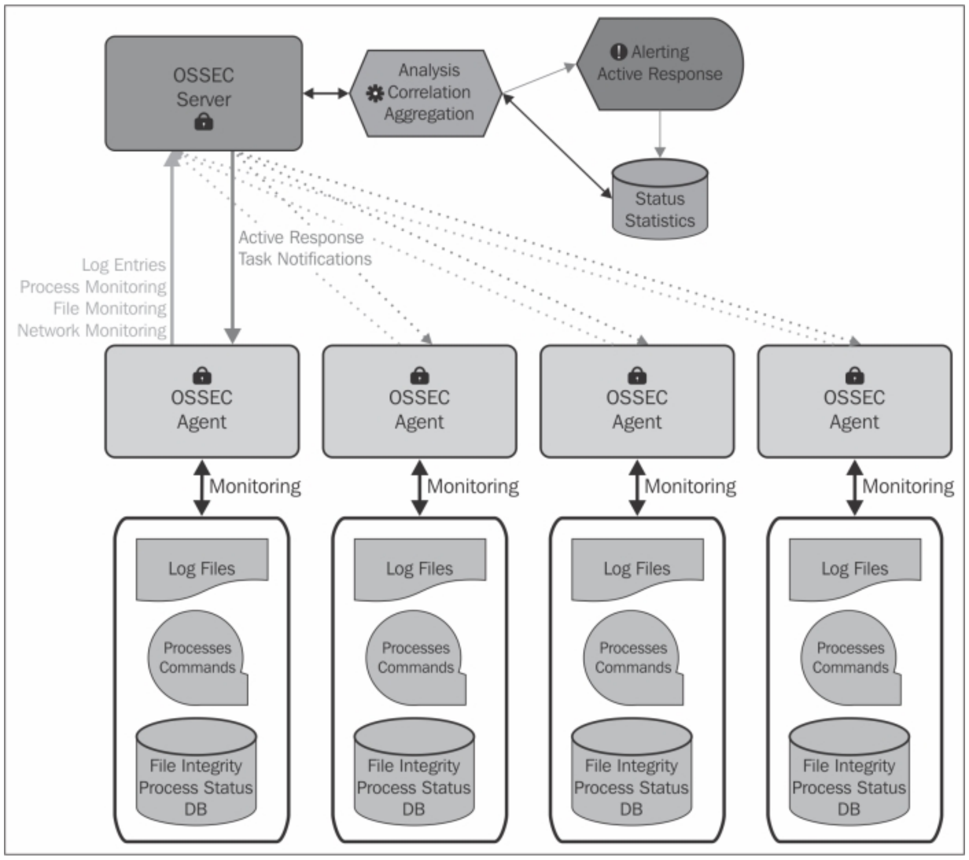


Figure 6. OSSEC architecture [4]

OSSEC has an extensive code-base with active community development. It comes with hundreds of log decoders in a default installation and is customizable to additional applications. Additionally, it offers the flexibility of “agentless” monitoring for systems such as routers, firewalls, etc. This feature allows it to stack intrusion detection power with other network-based products like SNORT while maintaining a centralized management architecture.

2.8 Advanced Message Queuing Protocol

Enterprise developers adopted Message Oriented Middleware (MOM) because of the desire for flexible communication while remaining reliable and secure. MOM allows efficient message delivery between applications on heterogeneous systems, effectively bypassing the expensive use of explicit connections like Virtual Private Networks (VPNs). Usually, this is accomplished with asynchronous message delivery through a queue framework directly on the MOM middleware. The Advanced Message Queuing Protocol (AMQP) aspires to define MOM, and has been widely adopted as an open standard for “business messaging” [44].

AMQP originated in 2006 due to the demand of interoperability within the financial services industry [45]. It was developed as a binary, wire-level protocol, meaning the message format is strictly machine-readable as a stream of octets [46]. Its ubiquitous and pervasive nature allows AMQP to be highly scalable, proving itself as a high performance, fault-tolerant, and lossless messaging infrastructure. Security and message integrity is implemented through mutual authentication and the Transport Layer Security (TLS) encryption scheme. Figure 7 displays how AMQP relies upon a publish-and-subscribe model, consisting of three primary components:

- **Publishers:** Clients generating the messages
- **Broker:** A server or daemon program containing an exchange, one or more routing engines, and message queues

- **Consumers:** Clients consuming one or more messages from the queue(s) on the broker

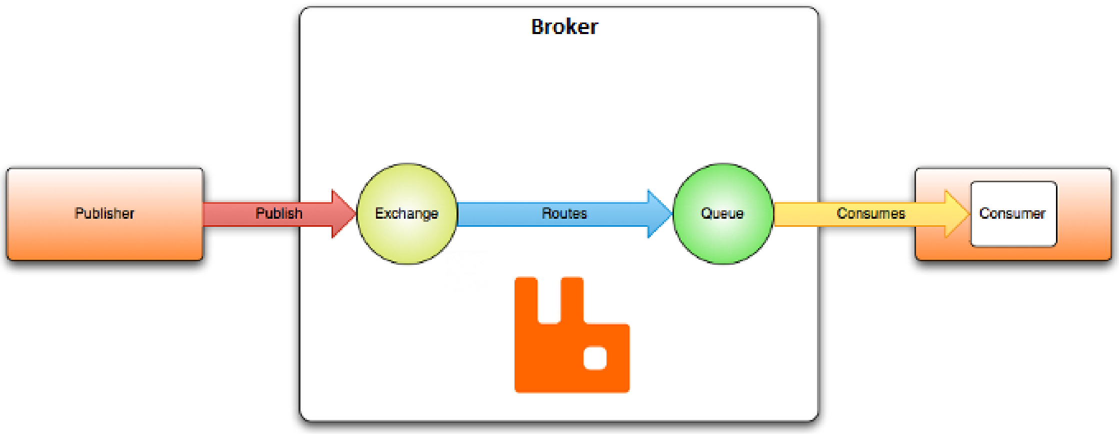


Figure 7. AMQP architecture [5]

2.8.1 Brokers.

The broker's role is to receive messages from the publisher applications and route them to the appropriate consumer. This transfer is first accomplished using an exchange module, which acts as a mailbox for any incoming messages. Exchanges then copy and distribute the message to outbound queues using binding rules. The actual delivery is initiated by the broker if the consumer is subscribed to the queue, or consumers can fetch/pull messages at their discretion.

As a programmable protocol, AMQP routing is defined at the application layer, then executed at the network layer. As a result, it is possible to build a scalable broker topology using a k-nomial tree scheme, varying in both depth and breadth [47]. While this only applies to a disjoint federation architecture, custom topologies can meet the routing needs of the target system.

2.8.2 RabbitMQ.

RabbitMQ is an open-source middleware, specifically acting as a message broker and queuing server. It supports many messaging protocols, including AMQP. In a multi-broker model, RabbitMQ can either be clustered together (forming a single, logical broker) or loosely connected through federation or tree structures. Additionally, it is officially supported on several operating systems and programming languages, including Java, Python, and C/C++ [5].

2.9 Related Research

This research aims to assist in the intrusion response process by utilizing SDN and HIDS together in a harmonious system with efficient message passing. Figure 8 illustrates Michael Todd's Dynamic Security Control using SDN (DSCS) system proposal and how these can work together to provide an immediate response to actively quarantine potential intrusions. HIDS monitors the critical end systems, generating alerts which are passed to the AMQP Broker. A SDN controller subscribes to these alerts, and a network application interacts with the Northbound API in response to the alert. All of these components working together results in dynamic and automated flow table updates which monitor, isolate, and block security policy violations.

The limitations of Todd's previous work have been quantified, and this research seeks to mitigate shortcomings and expand functionality in the following ways:

- The use of Floodlight's stateless firewall could not interrupt active connections if they were found malicious. This research seeks a viable solution.

- The DSCS system was not interfaced with an actual IDS. This research seeks to use industry standard software (e.g., OSSEC) and its APIs to generate alerts.

- A database schema could be incorporated to allow alert sharing and archival tracking, meeting standard logging policy requirements and improving scalability.

- Different approaches to intrusion response (firewall updates, first hop modifications, honeypot redirection, etc.) can be tested for performance, efficiency, and/or latency.
- The AMQP publisher/broker model can be replaced with a similar messaging design within a Security Incident Management (SIM) or Security Information and Event Management (SIEM) product.
- The programmability of the AMQP protocol can be replicated in a SDN environment because routing decisions are defined outside the network layer of packet forwarding devices.

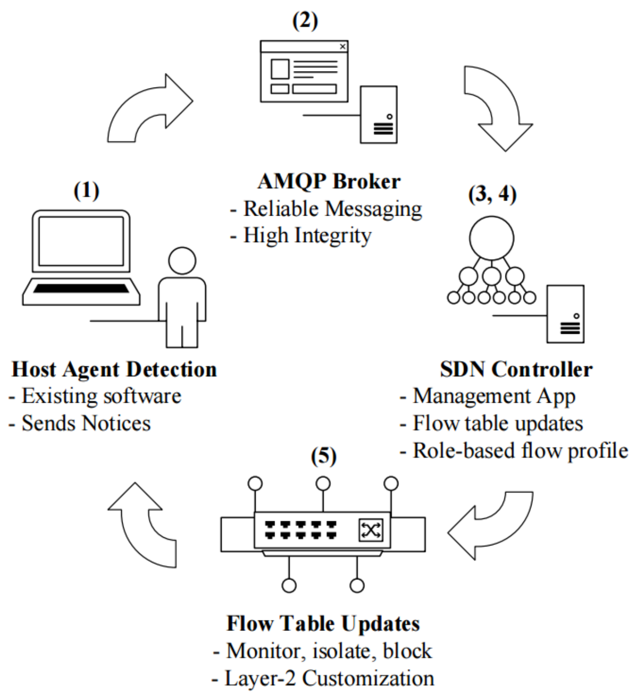


Figure 8. Dynamic Security Control using SDN (DSCS) process [6]

2.10 Conclusion

This chapter explains SDN's differences between basic networking and its historical development. It describes the fundamental components of SDN's framework, and presents use cases of how it can help improve network capabilities. Lastly, it explores both the security liabilities and potential security applications that SDN could provide. The critical, legacy-style role of network architecture may make SDN a slow adopting technology, but the flexibility of dynamic programming proves itself priceless.

III. AHNSR Design

3.1 Overview

This research introduces the Active Host-based Network Security Response (AHNSR) framework, a redesign of the DSCS process discussed in Section 2.9. This chapter provides a detailed description of each component in the AHNSR system and its role in the experiment. In understanding the key technologies in SDN, the four main components of this new system under test can be defined: agent, OSSEC, flow profile, and the SDN controller. The interface between OSSEC and the SDN controller is a novel contribution that enables a well-vetted, open-source HIDS platform for generating security events. A testable SDN-controlled network is constructed with multiple hosts, OpenFlow enabled switches, and a Floodlight controller, all linked to the novel interface for the HIDS framework.

3.2 System Summary

Figure 9 displays a simplified network diagram of all components described in the AHNSR design: SDN controller, OSSEC server, switches, and workstation agents. The boundaries are limited to these components, and do not include other services normally present in an enterprise network. However, as discussed later in Chapter 4, the test network does attempt to more accurately represent a real-world network by utilizing physical switches instead of virtual switches. Therefore, all Ethernet communication (including OpenFlow) between any two hosts are physically transmitted through independent network interface cards instead of residing within the same virtual bridge.

Figure 10 provides a high-level view of the AHNSR system, describing the interaction between each component. The process follows this cycle:

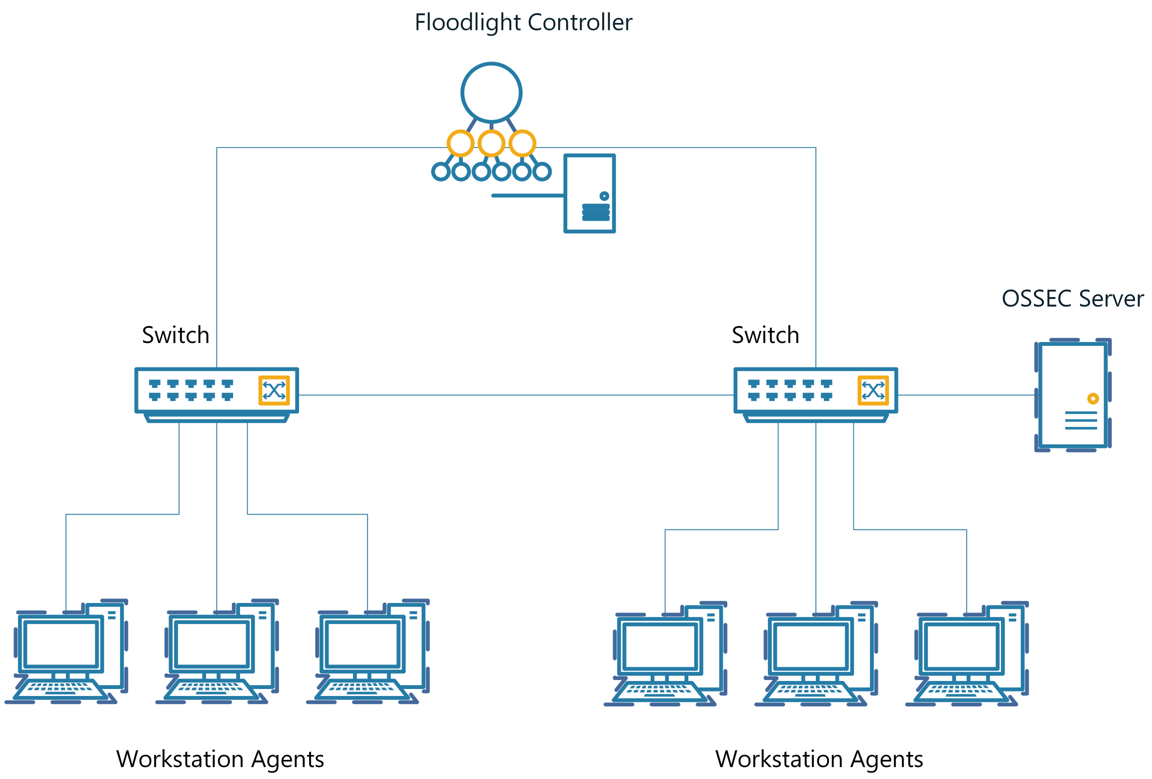


Figure 9. Network diagram of AHNSR components

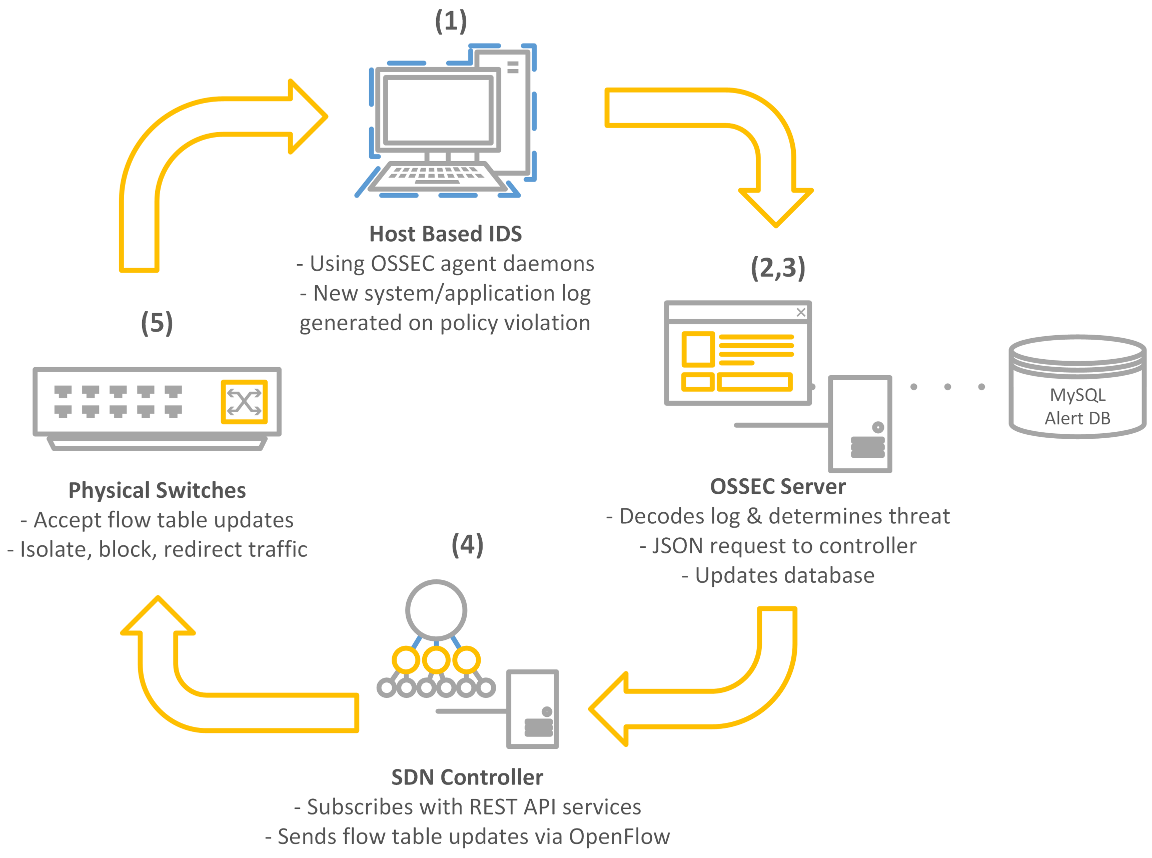
1. Individual agents register and connect with the OSSEC server during deployment/configuration in a Software-Defined Network. While connected, OSSEC continually monitors the agents using multi-log analysis, rootkit detection, and file integrity checking. Based on rule-sets and threshold levels that represent the enterprise IT policy, security alerts are generated when an event occurs on any monitored agent.

2. The OSSEC server decodes, analyzes, and determines the appropriate active response to protect the network.

3. A custom active-response script then interacts with a secure REpresentational State Transfer (REST) API of the SDN controller in order to request flow entries be pushed to network switches.

4. The SDN controller then communicates via OpenFlow channels to the physical switches

5. These switches, in turn, accept the flow tables updates which ultimately isolate, block, or redirect traffic.



(1)
Host Based IDS
- Using OSSEC agent daemons
- New system/application log generated on policy violation
(2,3)
MySQL Alert DB
OSSEC Server
- Decodes log & determines threat
- JSON request to controller
- Updates database
(4)
SDN Controller
- Subscribes with REST API services
- Sends flow table updates via OpenFlow
(5)
Physical Switches
- Accept flow table updates
- Isolate, block, redirect traffic

Figure 10. AHNSR system design

3.3 SDN Controller

The SDN controller is implemented using the master branch of open-source Floodlight project (commits made up to 23 September 2016), using a Java 1.8 compatible update. Figure 11 shows the relationship among the Floodlight controller, the applications built as Java modules within Floodlight, and the applications built over the Floodlight REST API [7]. Internally, the Floodlight controller contains a set of common functionalities that control and inquire about an OpenFlow network, while applications on top of it utilize different features to solve user needs throughout the network.

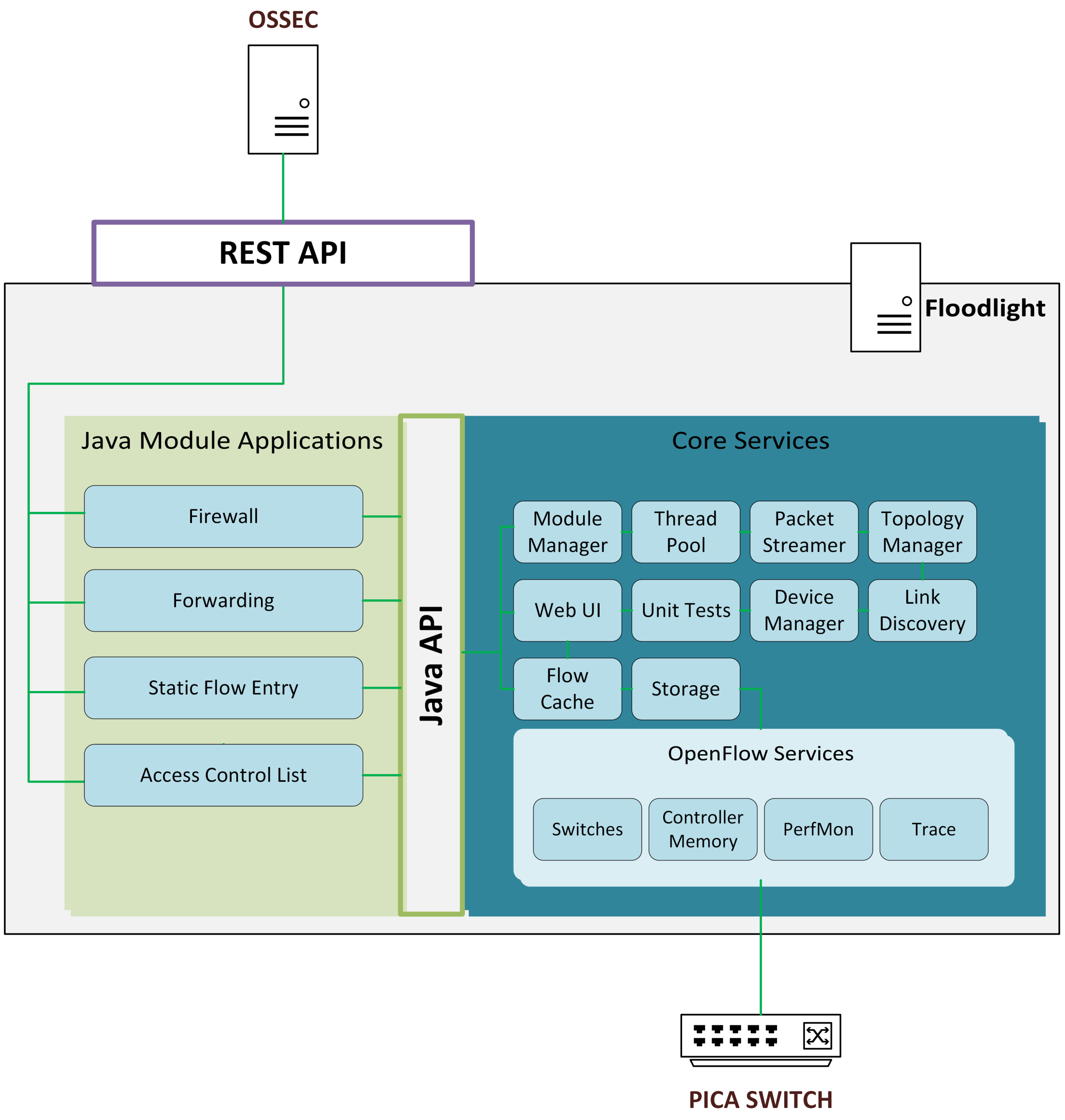


Figure 11. Floodlight architecture and module interaction [7]

Upon execution, Floodlight and its set of Java module applications (those loaded in the Floodlight properties file) begin their services. The REST APIs exposed by all running modules are available via the specified REST port (8080 and 8081 for HTTP/HTTPS, respectively). These modules allow the normally-abstracted OSSEC server to also function as a pseudo-REST application, and can retrieve information by invoking services through JavaScript Object Notation (JSON) constructed commands/requests sent to the controller REST port via HTTP/HTTPS. This request is demonstrated in Section 3.4.4, OSSEC's SDN response.

This research selectively utilizes Floodlight's `StaticFlowEntryPusher`, `ACL` (Access Control List) and `Firewall` Java module applications. These modules allow the AHNSR system to control flows at a very fine level, bypassing some limitations in previous related work. For example, active malicious connections can be immediately terminated by forcing the switch to create a new static flow, consequentially dropping those packets. However, these modules operate differently – the Firewall module uses reactive flow entries while the ACL and Static Flow Entry modules use proactive flow entries.

3.3.1 Reactive/Proactive Flow Entry.

Figure 12 illustrates the difference between reactive and proactive operations. The Firewall module is designed reactively, meaning the switch must ask the controller how to handle new packets. After receiving the switch's OpenFlow PACKET_IN message, the controller triggers the Firewall module. This module operates by comparing each incoming PACKET_IN message against its pre-configured allow/deny list from the highest priority until either a match is found or the list is exhausted. If a match is found, the rule's action (ALLOW or DENY) is stored in a IRoutingDecision object to be passed on to the rest of the packet-in processing pipeline, which normally ends with the default Forwarding module. The forwarding module then reactively inserts flows in switches, effectively routing packets from the source to its destination.

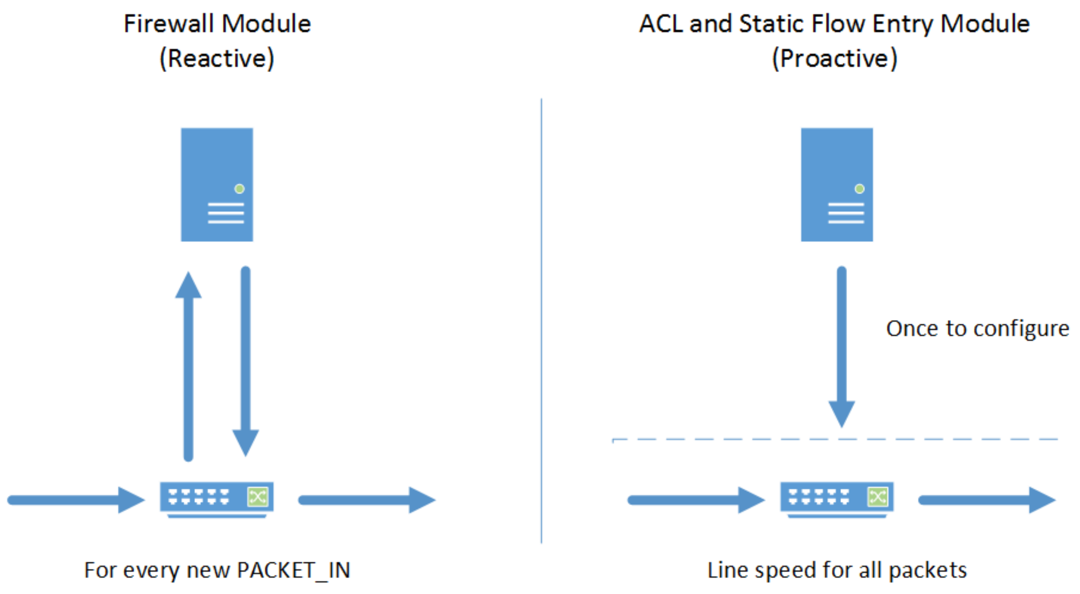


Figure 12. Reactive vs. proactive Floodlight modules

Alternatively, proactive flow entries are sent by the controller *before* the switch handles incoming packets. This requires a single OpenFlow FLOW_MOD message – then all subsequent matching flows are automatically forwarded at line speed.

Both reactive and proactive methods are used in this research as they are independently executed by the Java application modules.

3.4 OSSEC Configuration

OSSEC is installed in a server/agent architecture, where agents communicate with a server for centralized analysis. The server and Linux agents are running OSSEC's pre-release of version 2.9 (v2.9.0 RC3), while the Windows workstations are running OSSEC version 2.8.3 (the latest version available for Windows clients). Figure 13 provides a high-level view of the agent/server log flow and the various processes handling each action:

1. Agents are responsible for log collecting using the *ossec-logcollector* process. New log entries are forwarded to the server securely via UDP port 1514 using

messages encrypted with the blowfish algorithm and compressed using zlib [48]. This network communication is handled by the *agentd* and *remoted* processes, located agent-side and server-side respectively.

2. The server's *ossec-analysisd* process decodes the log using regex parameters, determines the format, then matches it to any defined rules.

3. The server's *ossec-dbd* process is responsible for storing alerts into a database, while the *ossec-execd* process executes any active-response actions that may be necessary for the corresponding alert.

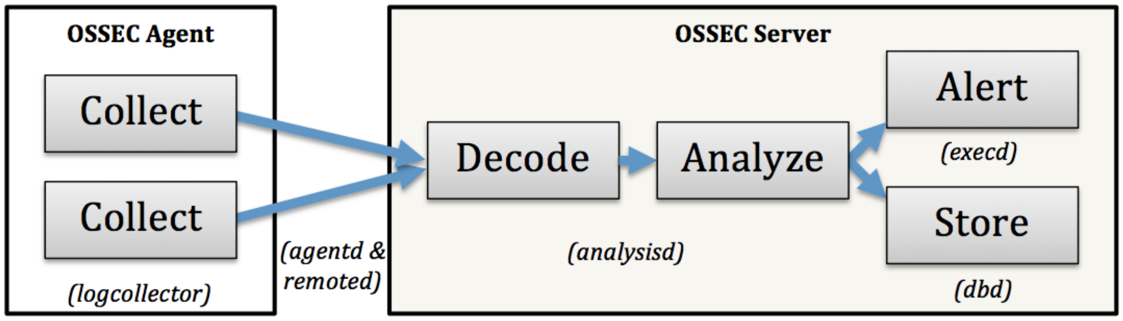


Figure 13. OSSEC log flow in a server/agent architecture

3.4.1 Agents.

Since the agents are responsible for monitoring log files, the experiment uses a simplified, consolidated adaptation to simulate security events logged to various places on the workstation. During configuration, an empty plain-text file is created on each agent, representing a no-noise syslog file. In order to have each agent's *ossec-logcollector* process actively monitor the experimentation log, this research includes a *localfile* element pointing to the local experimentation log. The format is specified as *syslog* – a generic format which parses one log per line.

3.4.2 Decoder/Rules.

Log rules are stored internally on the server in an eXtensible Markup Language (XML) tree structure, independent of the initial log format. For AHNSR experiments, this research constructs a custom decoder and ruleset to simulate security events by adding the following construct to */var/ossec/etc/decoder.xml* on the OSSEC server:

```
<decoder name="experiment1">
        <prematch>^experiment1</prematch>
</decoder>
<decoder name="experiment-alert">
        <parent>experiment1</parent>
        <regex offset="after_parent">^an event occurred. LEVEL=(\d+)
        ID=(\d+)</regex>
        <order>extra_data, id</order>
</decoder>
```

This decoder successfully consumes any logs with the pattern "experiment1: an event occurred. LEVEL=# ID=#" Decoders provide a way to extract field data from the message to be used in context - this decoder utilizes this by extracting the level information and storing it as *extra_data* variable. The fields are described as follows:

- **Level:** This is a threat severity level index used internally in the OSSEC IDS framework. Every rule is classified from the lowest threat (00) to the maximum level (15). The descriptions for each level are provided in Appendix M.

- **Log ID:** This is the log ID generated by the agents when they write a new line to the log file. It is used for message tracking during the tests.

- **Alert ID:** This is a unique identifier assigned by OSSEC for every alert it generates. The number also acts as the primary key for the alert database described in Section 3.4.5.

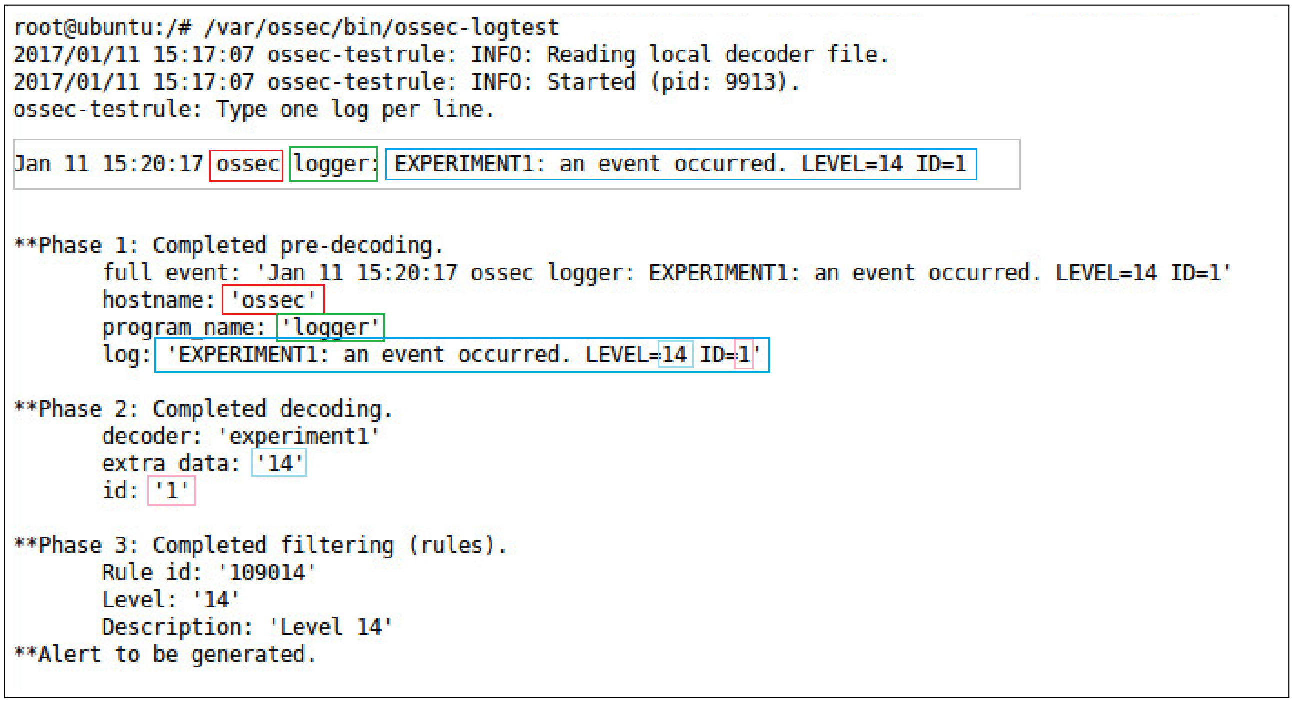
Once a log is decoded to a specific format, rules can be applied toward the extracted information. For example, a Secure Shell (SSH) rule may extract the source IP address when a decoder identifies the format of a failed authentication log - which means it can use that information in generating an informative alert for administrative action. For this experiment, decoding should match the simulated security level to the alert level, so the following rules are constructed on the OSSEC server, in the */var/ossec/rules/ossec_rules.xml* file like so:

```
<group name="experiment1">
        <rule id="109000" level="0">
                <decoded_as>experiment1</decoded_as>
                <description>custom experiment alert</description>
        </rule>
        <!-- Alert for level 1 -->
        <rule id="109001" level="1">
                <if_sid>109000</if_sid>
                <extra_data>1</extra_data>
                <description>Level 1</description>
        </rule>
        <!-- Alert for level 2 -->
        <rule id="109002" level="2">
                <if_sid>109000</if_sid>
                <extra_data>2</extra_data>
                <description>Level 2</description>
        </rule>
        ...
        <!-- follow for levels 3 to 15 -->
        ...
</group>
```

A parent group is established to hold all related rules. As a "catch-all" bucket, if a log is decoded as the experiment format, it is first flagged to match rule 109000. This initial match immediately eliminates all other rule formats except those with a corresponding *if_sid* element. The *if_sid* field requires a rule to be flagged as another rule first, in this case rule ID 109000. Additional rule-level combinations can be included for increasing level values. This way, the system can efficiently emulate rule threat level promotion or demotion depending on the context of the actual log.

For example, Figure 14 shows OSSEC's log-test output, a testing/verification tool that decodes exactly how the real *ossec-analysisd* process does. This illustrates three phases when decoding the sample log:

```
Jan 11 15:20:17 ossec logger: EXPERIMENT1: an event occurred. LEVEL=14 ID=1
```



```
root@ubuntu:/# /var/ossec/bin/ossec-logtest
2017/01/11 15:17:07 ossec-testrule: INFO: Reading local decoder file.
2017/01/11 15:17:07 ossec-testrule: INFO: Started (pid: 9913).
ossec-testrule: Type one log per line.

Jan 11 15:20:17 ossec logger: EXPERIMENT1: an event occurred. LEVEL=14 ID=1


**Phase 1: Completed pre-decoding.
       full event: 'Jan 11 15:20:17 ossec logger: EXPERIMENT1: an event occurred. LEVEL=14 ID=1'
       hostname: 'ossec'
       program_name: 'logger'
       log: 'EXPERIMENT1: an event occurred. LEVEL=14 ID=1'

**Phase 2: Completed decoding.
       decoder: 'experiment1'
       extra_data: '14'
       id: '1'

**Phase 3: Completed filtering (rules).
       Rule id: '109014'
       Level: '14'
       Description: 'Level 14'
**Alert to be generated.
```

Figure 14. Decoding phases for sample log entry

- **Phase 1:** The text is recognized as a generic syslog entry, which follows this format: [`<date> <time> <hostname> <program name>: <--log-->`]. Therefore, it first pulls off the hostname and program name fields.

- **Phase 2:** OSSEC finds the correct decoder for the program's actual log entry. In this case, the custom *experiment1* decoder is selected due to the regex pre-match on the text "experiment1". This decoder also recognizes the content of the two other fields, *extra_data* (the visible place-holder for the alert level), and *id* (Log ID, not Alert ID).

- **Phase 3:** A rule assignment is the last step. OSSEC searches the XML tree, and can match on rule 109014 as the following elements were true:
  `<decoded_as> = experiment1` and `<extra_data> = 14`
  Addtionally, this rule ID has a threat severity level of 14, matching the log-test output in Figure 14.

3.4.3 Active Response.

Active response is technically separated into two elements within the OSSEC system: commands and configuration. A command simply provides a symbolic link to the actual executable and can specify expected parameters for a given script. This experiment defines the SDN-response command in */var/ossec/etc/ossec.conf* on the OSSEC server:

```
<command>
	<name>SDN-response</name>
	<executable>SDN-response.sh</executable>
	<expect></expect>
</command>
```

While the executable does expect the alert level value (extracted as *extra_data* during decoding), the expect tags are left empty here because the only valid options are *srcip* and *username*. The SDN-response script can still pull the value through the default parameters forwarded, as shown in the Section 3.4.4.

However, the command alone is not enough to define an active response. In the configuration (Appendix C), the commands are binded to events defined within their own active-response element as shown:

```
<active-response>
	<command>SDN-response</command>
	<location>server</location>
	<rules_group>experiment1</rules_group>
</active-response>
```

The command is used to link the response to the executable's symbolic link above. *Location* is where the command should be executed (alerts can be designed to trigger executables residing on either the server or the agents themselves). Lastly, the *rules_group* is essential, as any alert generated from this rules group will execute the response. The experiment has all rules defined under the experiment1 group in the rules.xml described in Section 3.4.2; thus, the single group identifier can be referenced here.

3.4.4 SDN Response.

The active response scripts utilize the RESTful interface to the SDN controller in order to request changes to the network. The essential core curl command format in the scripts is:

```
curl -X POST -d '{
        "switch": "00:00:00:00:00:00:00:01",
        "name":"flow-mod-1",
        "cookie":"0",
        "priority":"1",
        "eth_type":"0x0800",
        "in_port":"1",
        "active":"true",
        "actions":"output=2"
        }'
http://10.231.0.10:8080/wm/staticflowpusher/json
```

This constructs a JSON string, and sends it to the controller (at IP=10.231.0.10 in this experiment) on the port where its REST services are running (8080) via HTTP POST. This particular request is sent to the `StaticFlowEntryPusher` module, which allows a user to manually insert proactive flows and groups into an OpenFlow network. The command above requests a new flow for a particular switch, which applies to all packets with eth_type of 0x0800. This relates to the two-octet `EtherType` field in an Ethernet frame, of which 0x0800 defines IPv4. In summary, this request generates a new flow on the switch with a Media Access Control (MAC) address of

[00:00:00:00:00:00:01], directing all IPv4 traffic coming into port 1 to output on port 2. The full *SDN-response.sh* script is included in Appendix F.

Normally, switches must request how to handle new traffic via a `PACKET_IN` OpenFlow message sent to the controller – but as a proactive flow, no additional requests are required to handle subsequent traffic that matches this flow.

3.4.5 Database.

OSSEC database output is enabled with a *make* command during OSSEC server compiling and installation. The database is managed through the *ossec-dbd* process on the server, which inserts logs, agent and server information into a MySQL database on the same machine. This schema's Enhanced Entity Relationship (EER) diagram is shown in Figure 15, illustrating the system properties and relationships.

Notable mentions in the EER diagram include:

- The *alert* table includes the rule_id associated with the alert.
- The *alert* table contains foreign keys to the location where the log originated, as well as the server ID handling the alert.
- Timestamps upon entry addition are included, which can be used to validate OSSEC processing when comparing to the log generation timestamp.

3.5 Security

AHNSR is a privileged system with unique network control, and therefore requires a strong security design. The attack vectors discussed in Section 2.5.1 reveal how the STRIDE methodology (Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of privilege) relate to any Software-Defined Network. This research helps implement these security properties as follows:

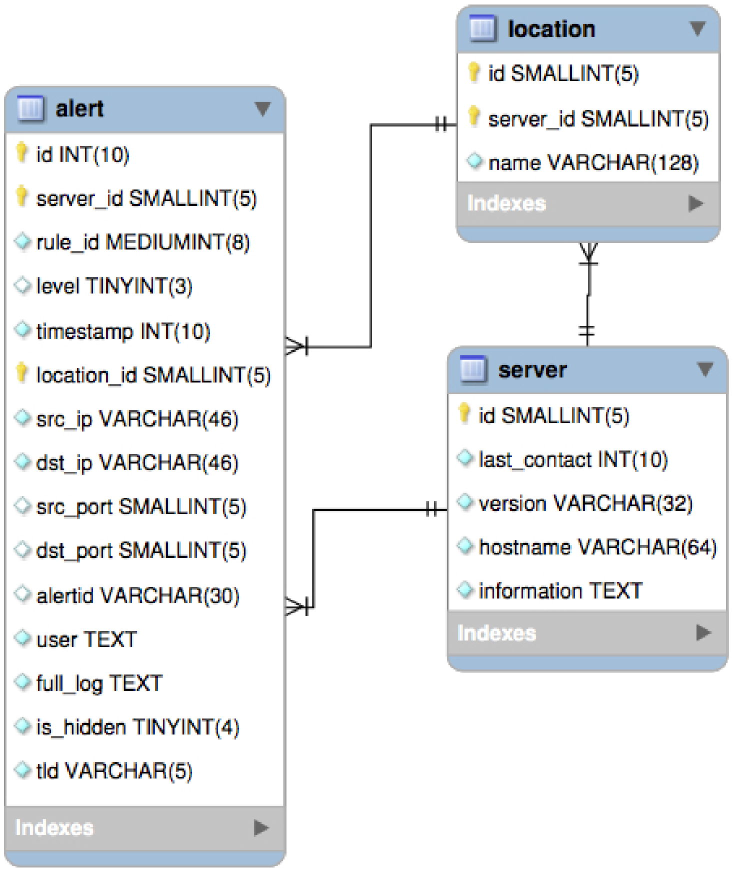


Figure 15. EER diagram of OSSEC database

- **Authentication**: All critical systems are fully authenticated by standard PKI (Public Key Infrastructure) using Secure Sockets Layer (SSL) encryption. Private keys are generated on the Pica switch, Floodlight, and OSSEC using the native OpenSSL library, and their self-signed certificates are manually installed on the corresponding partner systems. The Floodlight controller is configured to support only trust-based SSL for both its OpenFlow connections and REST API server. These channels operate independently with separate Java keystores, so settings in one module do not impact the other (Appendix E). As a result, all *curl* requests to Floodlight by OSSEC must include this custom public/private key format, and redirected to the secure HTTPS port:

```
curl
        --cacert /rest-cacert.pem
        --cert /ossec-selfsigned.crt
        --key /ossec-selfsigned.key
        -X POST -d '{
                "switch": "00:00:00:00:00:00:00:01",
                "name":"flow-mod-1",
                "cookie":"0",
                "priority":"1",
                "eth_type":"0x0800",
                "in_port":"1",
                "active":"true",
                "actions":"output=2"
        }'
https://10.231.0.10:8081/wm/staticflowpusher/json
```

- **Confidentiality**: OpenFlow TCP traffic is executed through "out-of-band" management, meaning a dedicated, isolated management port on each switch communicates directly to Floodlight, and therefore sniffing is not possible without switch access. Similarly, a dedicated bridge on the server allows OSSEC to communicate with Floodlight on a dedicated channel. However, by nature of the authentication mechanisms above, OpenFlow traffic is encrypted via SSL Public Key Infrastructure (PKI) and communication to the REST server is encrypted via HTTPS. Furthermore, OSSEC host log messages are encrypted using the blowfish algorithm and compressed using zlib, utilizing private keys installed during initial host configuration.

- **Integrity**: By default, Floodlight confirms successful connections with switches using the Openflow 1.4 Hello sequence of HelloElements [49]. OSSEC also attempts to prevent falsification and replay attacks using message counters within the encrypted payload. All OSSEC configuration files on workstation hosts are also added to the syscheck real-time monitor list for immediate alerts on alteration.

- **Availability**: Floodlight is a robust controller that can maintain 100% uptime under normal load. Additionally, OSSEC has a real-time statistics module that learns average message load, and can trigger custom alerts when a threshold is crossed, possibly indicating a denial of service attack. Therefore, even if a compromised host attempts to flood OSSEC's UDP receipt queue on port 1514 with legitimate logs, an active response is configured to block the offending host if an "Excessive Events" rule fires.

3.6 Complete AHNSR Response Flow

Figure 16 illustrates the complete detailed AHNSR framework, putting together all the components described in this section. The green line represents the response flow between and within each component, starting at log generation on the agent. The new entry is collected by the *logcollector* daemon and forwarded to the OSSEC server through the *agentd/remoted* communication pipeline. OSSEC's three analysis phases are included and a matching rule results in both database archival and executing the *SDN-response.sh* script. This queries Floodlight's REST API, invoking services published by its Java modules applications. These modules interact with the core internal services responsible for OpenFlow communication to the switches. The request is received by the switch resulting in the successful flow table update.

3.7 Design Summary

This chapter describes the individual components of the AHNSR system. The design is an advanced approach to automate network reconfiguration in reaction to security policy violations at user workstations.

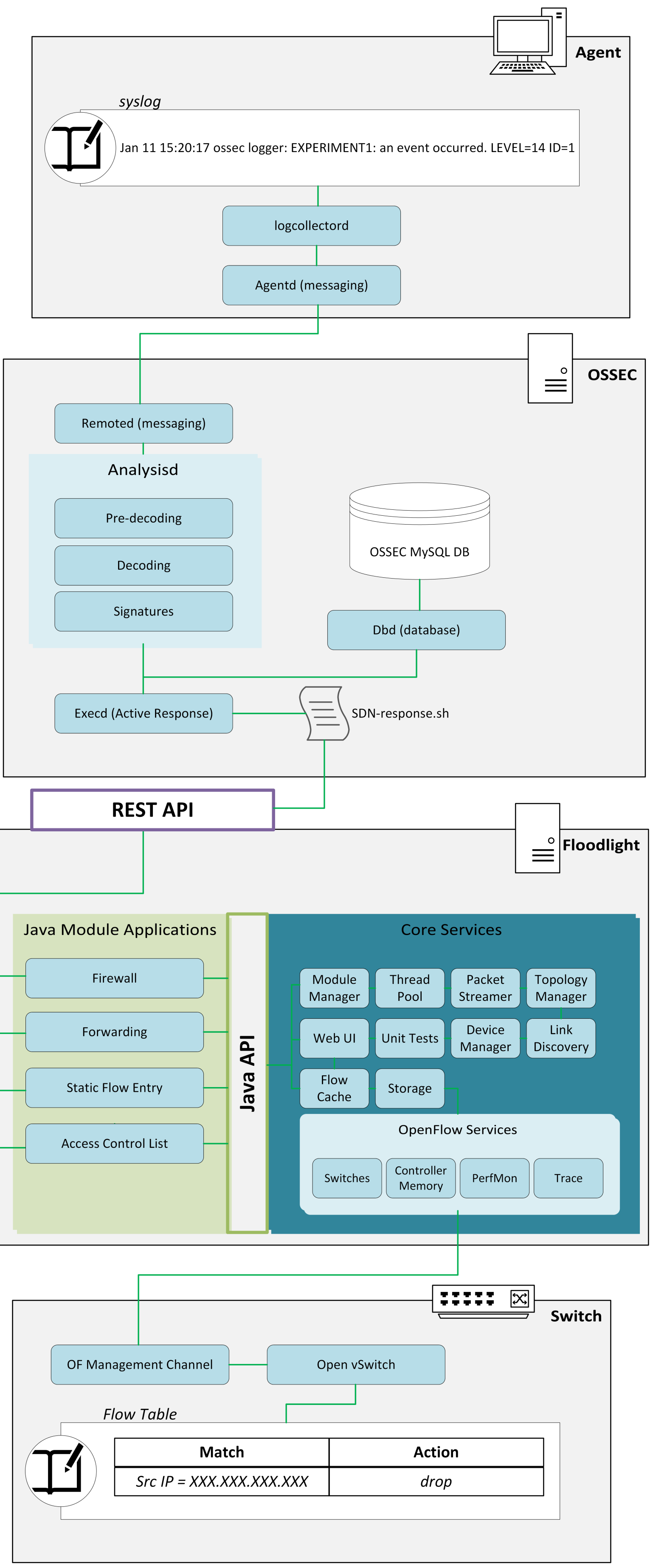


Figure 16. Response flow through agent, OSSEC, Floodlight, and switch

IV. Methodology

4.1 Problem/Objective

This research seeks to expand on Todd's Dynamic Security Control System (DSCS), a process of utilizing SDN with host-based security alerts [6]. The limitations of Todd's work in this area have been quantified, and this research seeks to mitigate shortcomings and expand functionality in the following ways:

Limitation Mitigations

- The use of Floodlight's stateless firewall could not interrupt active connections if they were found to be malicious. This research seeks a viable solution.
- The DSCS system was not interfaced with an actual IDS. This research seeks to use industry standard software (OSSEC) and its APIs to generate alerts.

Improvements

- A database schema is incorporated to allow alert sharing and archival tracking, meeting standard logging policy requirements and improving scalability.
- Different approaches to intrusion response (firewall or access control list updates, first hop modifications, honeypot redirection, etc.) are allowed for more dynamic control.
- The AMQP publisher/broker model can be replaced with a similar messaging design within a Security Incident Management (SIM) or Security Information and Event Management (SIEM) product.
- The programmability of the AMQP protocol can be replicated in a SDN environment because routing decisions are defined outside the network layer of packet forwarding devices.

After redesigning the process, this research introduces Active Host-based Network Security Response (AHNSR), which implements the functionalities listed above. Therefore, this experiment functions more as an evaluation, testing how the AHNSR system performs under both normal and stressed loads. The evaluation results will help inform administrators of optimal configuration settings and set a quantifiable upper limit for how many agents they can support. As an overall efficiency standard objective of maintaining a less than 2 second delay in responding to new log entries, the limit can be determined in statistical trends as network load increases.

4.2 System Under Test

Figure 17 displays the System Under Test (SUT) and Component Under Test (CUT) diagrams. The workload factors consist of SDN response method and an Event Per Second (EPS) level, described in Section 4.4. Section 4.5 discusses the computing parameters, which are held constant. The targeted components consist of the agents' log collection, OSSEC's analysis/execution performance, active response script, and Floodlight's REST API. Recorded metrics, described in Section 4.3, consist of Central Processing Unit (CPU) utilization, response time, and the percentage of successful alerts generated.

Agents and servers are segmented on a SuperMicro SuperServer X9QR7-TF+ with 16 Xeon E5-4610 v2 processors, eight 1000BASE-T network interface cards, and 512GB of RAM. Figure 18 shows how the server (labeled as Aberdeen2) contains the following guest Virtual Machine (VM)s: Linux Ubuntu 14.04 guests are assigned two cores, 8GB of RAM, and run the OSSEC Agent process to register and connect as agents to the OSSEC server. Floodlight runs on a XUbuntu virtual machine with 8GB RAM and is assigned two processors. The OSSEC server runs on Ubuntu 14.04 64-bit Server (Linux 4.2.0-27), also with 8GB RAM and is assigned two processors.

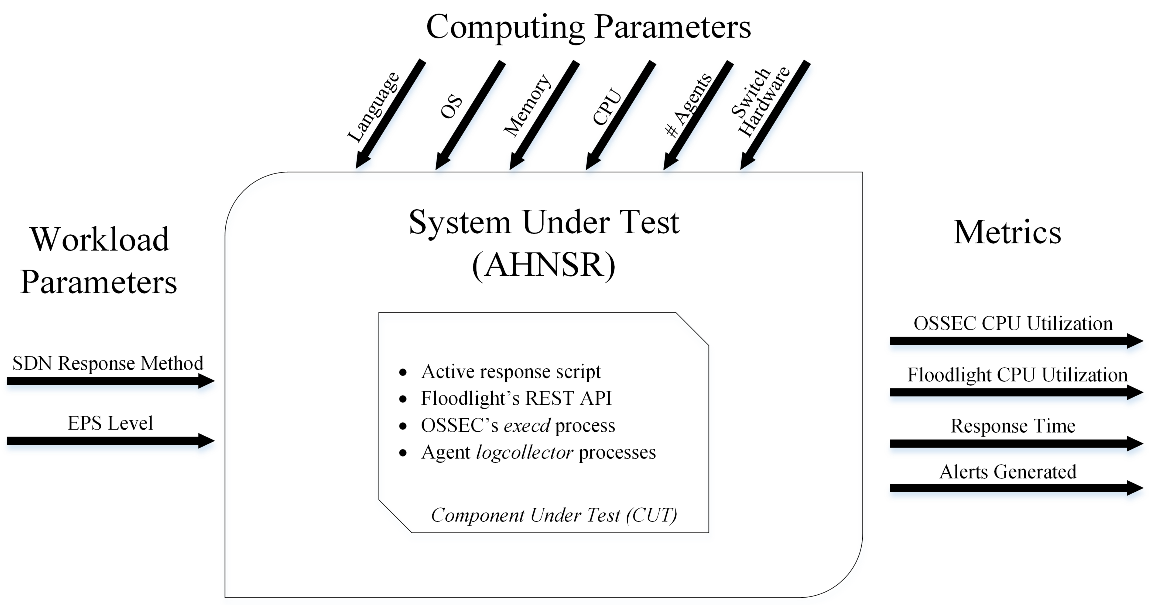


Figure 17. System Under Test (SUT) and Component Under Test (CUT)

All guests have shared access to 1000BASE-T Network Interface Card (NIC)s on a dedicated virtual bridge and all traffic is switched through a PICA manufactured switch. The PICA switch is a P3290 model, updated with the latest PicOS (2.7.1) with OpenFlow 1.4 support. The Floodlight controller utilizes out-of-band management to the PICA switch on an independent subnet (10.231.255.255), essentially providing a dedicated OpenFlow channel.

4.2.1 Assumptions.

The following assumptions are understood when designing and executing experiments for the AHNSR system:

1. The framework is dependent on well-configured host auditing and logging policies, as well as appropriate rule definitions on the management server.

2. Triggers and definitions are limited by the features provided by OSSEC version 2.9 RC3. More specifically, alert triggers are limited to system log analysis,

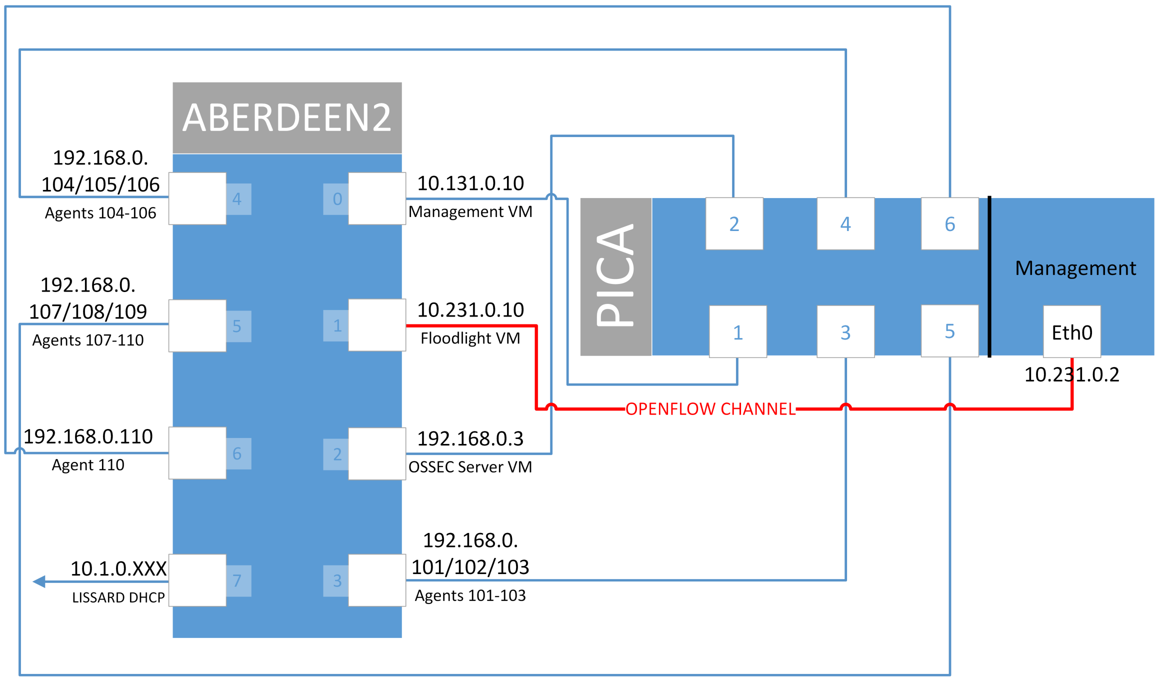


Figure 18. Advanced network diagram between server and switch

application log analysis, connection status, rootkit signature detection, selective registry changes, and/or selective file changes.

3. Rule definitions are limited to the following: alert level, specific rule ID, event frequency, maximum size, time frame, category, source IP, destination IP, username, hostname, program name, URL, and/or matching regex parameters.

4. The Floodlight controller has out-of-band management to all switches in the network. If this is not true, it does not ensure dedicated OpenFlow channels that may become congested otherwise, which can invalidate the observed results.

5. All switches must support OpenFlow 1.3 or higher, as some response modules require features in these versions.

6. A pre-configured flow is utilized for proactive forwarding to the IDS server. This is important for the UDP log message forwarding between the agents and server.

7. The OSSEC server can connect to Floodlight through a physical or virtual bridge. This research uses virtual switching with VMware's ESXi hypervisor.

4.3 Metrics

The stated system objectives are the primary influence in determining suitable performance metrics. As such, there are three main questions that help consider the overall performance of the AHNSR system in any given state. In relation to each question, a metric (or response variable) is adopted for experimentation:

1. As workload increases, are any new logs failing to be processed?

   Metric - **Alerts Generated (AG)**: Since the incoming logs are controlled, and each log should generate an alert, the AG metric can accurately measure if any logs were not processed. The AG metric can be expressed as the simple ratio measurement

   $$AG = \frac{SR}{LG} \tag{1}$$

   where LG represents the number of logs generated and SR represents the number of successful responses registered by the AHNSR system.

2. How long does it take for AHNSR to reconfigure the network through flow table updates?

   Metric - **Response Time (RT)**: The time from event generation to flow table update. RT requires high resolution timing, and its specific measurement implementation is discussed in further detail in Section 4.6.5.

3. How much CPU utilization on the main server components (OSSEC and Floodlight) is occurring during a given workload?

Metric - **Central Processing Unit (CPU)**: Both servers allow the user to query the system monitoring services to retrieve and log current CPU utilization information. CPU utilization is measured using the Linux `mpstat` package, which provides reports on the global average activities among all processors [50]. Both OSSEC and Floodlight execute `# mpstat 1 10` upon each experiment's starting trigger, which provide ten reports at one second intervals of the system's combined user and system level utilization. Thus, the utilization metric can be expressed as

$$CPU = 100 - \%idle \tag{2}$$

where *%idle* is the percentage of time when the CPUs are idle and the system does not have an outstanding disk I/O request.

Table 1 defines each metric's units of measurement, accepted range value, and expected range value.

Table 1. Performance metrics

Metric	**Units**	**Accepted Range**	**Expected Value**
AG (Alerts Generated)	numeric	0 to n, $n = \#$ logs generated	n
RT (Response Time)	μs	0 to ∞	$< 2 \text{ x } 10^6\ \mu s$
CPU utilization	%	0 to 100	Servers: $< 80\%$

4.4 Experiment Factors

Table 2 describes the experiment factors that are considered while developing the AHNSR framework.

Table 2. Experiment factors

Factor	**Type**	**Description**
Event density	continuous	How many Events Per Second (EPS) are being generated
Log-Only active response status	binary	Status of log only active response on server (running/not running)
Firewall active response status	binary	Status of firewall active response on server (running/not running)
Access Control List active response status	binary	Status of access control list active response on server (running/not running)
Static Flow Entry active response status	binary	Status of static flow entry active response on server (running/not running)

Event density is the primary factor, and is the main treatment in the experiment. The binary parameters consist of the four main SDN response methods available by Floodlight. Each method helps meet specific data security compliance requirements, such as those outlined in PCI DSS 2/3.2 [10]. Additionally, the Access Control List/Static Flow/Firewall modules are all capable of denying network access, which is the principal incident response strategy for any intrusion. Therefore, this research includes them as test factors to evaluate the most efficient and effective active response method. Treatments described in Section 4.6 allow for any covariance analysis described in Section 4.7.

4.5 Experiment Parameters

Many computing parameters are held constant based on the experiment configuration. This includes the component operating systems, resources (memory, CPU, and disk space), script languages, and switch hardware. The networking configuration established in the ESXi hypervisor remains constant as well. The tests run with the same 10 agents (also with identical computing parameters) connected and registered to the OSSEC manager.

Additionally, file integrity checking and rootkit detection are sub-functions provided by OSSEC (as discussed in Section 2.7), which may increase noise, latency, and/or CPU utilization on the agents. Therefore, this research holds those processes to a constant running state, helping evaluate for a worst-case scenario when all OSSEC-provided processes are demanded to run with a heavy workload.

4.6 Experimental Design

4.6.1 Expected Load Level.

Log generation is the most significant contributer to system load, as each log must process through the OSSEC workflow. In estimating average log generation, Solarwinds describes a sample enterprise as 1000 employee endpoints, 10 switches/routers, 25 various servers, 4 firewalls, and 7 IPS/IDS [51]. Their estimate for this environment is a total of 363 EPS. Compensating for potential peak periods and configuration variety, this research suggests an expected load of 500 EPS under normal operating conditions for an average enterprise network.

It is also important to note the difference between basic alerts and "SDN Response Demanding" alerts. Basic alerts are logs ignored or acted on without interaction with the SDN controller. "SDN Response Demanding" alerts occur when OSSEC receives a log and determines it is important enough to request a network modification to the SDN controller. Because threshold levels, enterprise needs, and SDN incorporation all vary significantly, the experiment eliminates any gross estimations by assuming all logs are “SDN Response Demanding”.

4.6.2 Treatments.

The planned experiment treatments are enumerated in Table 3. The columns represent each of the three factors discussed earlier. Tests 1–4 evaluate the expected operational load level whereas Tests 5–32 evaluate performance under extra load. Before starting treatment tests, measurements are taken at a baseline control level to gather data without external influence involved. This allows a relative level for output metrics, especially the CPU utilization at a restful state.

Tests 5–32 provide data for evaluation against the alert density parameter. Grouping between [5–11], [12–18], [19–25], and [26–32] capture the effect of different active response methods at increasing density levels. Tests 1–4 also capture these effects at the expected normal operating load.

4.6.3 Testing Process.

Syslog events are simulated on the agents using a custom decoder and a consolidated, monitored log file. Individual log threat levels are randomized when generated, and processed through the AHNSR system accordingly. OSSEC’s MySQL database keeps records of all non-ignored alerts, and active connection termination is enabled through the manual `StaticFlowEntry` module. Timing processes are built into the experiment at the agent (beginning), OSSEC server (middle), and Floodlight (end) to provide automated, high resolution data of the primary response variable, RT. Experiments are synchronously initialized using scripts across all agents.

Table 3. Treatments

Test #	Events Per Second (EPS) Levels	Log-Only	Firewall	Access Control List	Static Flow Entry
(control)	0	off	off	off	off
1	500	on	off	off	off
2	500	off	on	off	off
3	500	off	off	on	off
4	500	off	off	off	on
5-11	10, 100, 1000, 2500, 5000, 7500, 10000	on	off	off	off
12-18	10, 100, 1000, 2500, 5000, 7500, 10000	off	on	off	off
19-25	10, 100, 1000, 2500, 5000, 7500, 10000	off	off	on	off
26-32	10, 100, 1000, 2500, 5000, 7500, 10000	off	off	off	on

All experiments follow an identical testing process using automated scripts where noted:

1. First, Floodlight is restarted. This clears out all flows, establishes a baseline Java Virtual Machine (JVM) environment, and ensures connectivity with the OpenFlow enabled switch.

2. The OSSEC server is restarted, resetting its statistics modules. An additional reset script forces the server to clear its alert and active response logs.

3. The management *start-experiment.sh* script (Appendix B) is executed with arguments that specify the test configuration (e.g., event density, response method, number of trials, and number of agents). This script automatically

calls and manages the following steps (4 through 10).

4. An agent configuration bash script (*startLogger.sh*) is passed to all agents via listening Ncat sockets. Agents listen using `# ncat -l -k 12346 > listen.sh` and the management VM pushes the configuration using `# ncat 192.168.0.1XX 12346 < clientCommand.sh` where *clientCommand.sh* contains a call to the local log generation script with an EPS parameter.

5. The management VM pings all servers and agents to establish temporary flow entries. This ensures proactive forwarding and a better synchronized start time for the networked trigger activation.

6. The experiment is triggered from the management VM via Ncat listeners on the OSSEC server and agents. Similar to step 4, agents have persistent listeners as `# ncat -l -k 12345 --sh-exec "./listen.sh"` – Ncat's *sh-exec* tag runs a command by passing a string to a system shell [52]. The management VM simply connects to all simultaneously by forking each as a background process: `# ncat 192.168.0.1XX 12345 &`

7. Once triggered, the agents begin generating the appropriate amount of logs using the log generation script (Appendix A).

8. After the experiment completes, the management VM collects the OSSEC alert and active response logs via Secure Copy (SCP).

9. Data is parsed and validated against the OSSEC database using *dataParser.java* (Appendix G).

10. A consolidated data file in Comma Separated Values (CSV) format is pushed to the results repository using *drive*, a terminal push/pull style Google Drive client [53].

4.6.4 Repeated Measures.

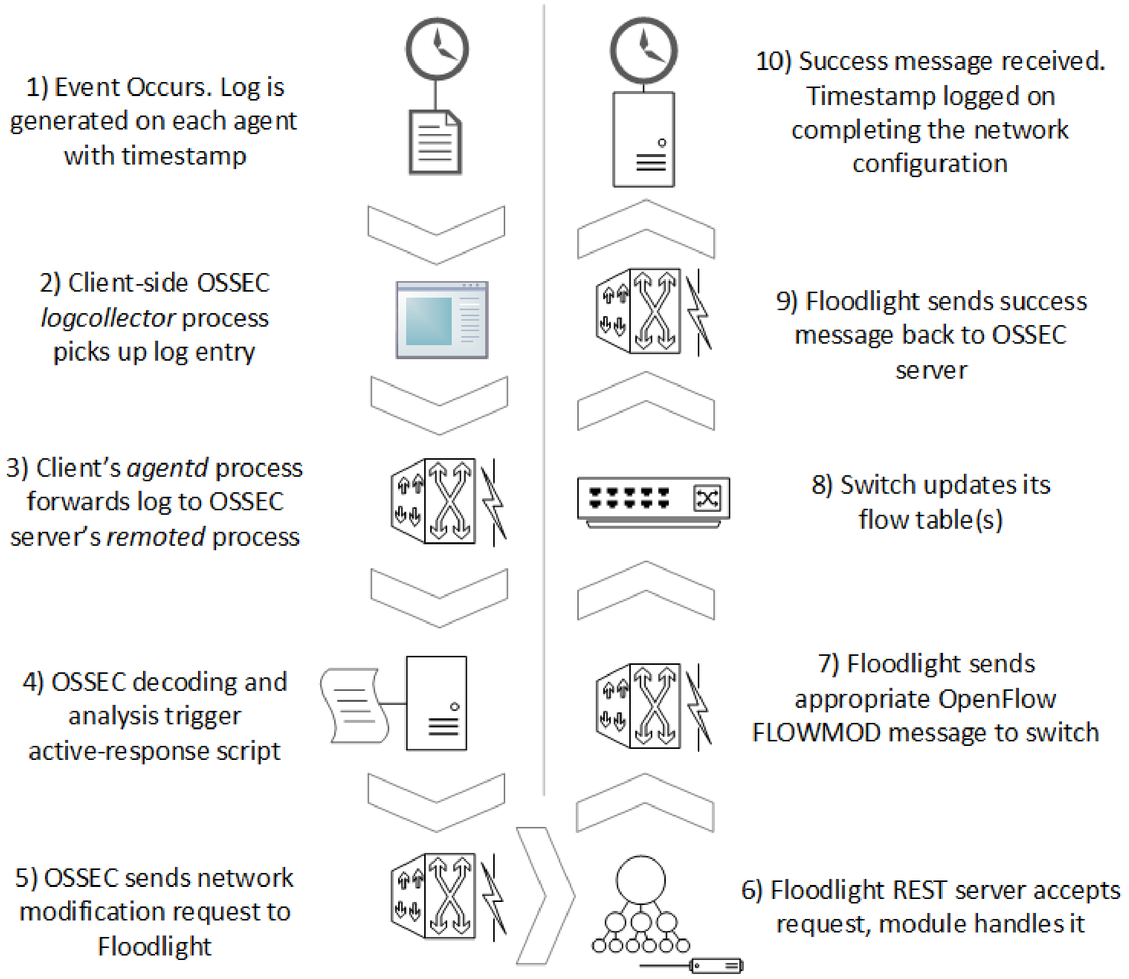
Each EPS level is equivalent to the aggregate log amount generated from all agents. For example, the 500 EPS level is the result of a 50 logs entered on 10 disparate agents. A two sample power t-test determines the appropriate number of samples (in aggregate) to collect in achieving a 95% confidence interval within ±.02 seconds. Using the highest standard deviation recorded during preliminary testing, .3667, the power test suggests collecting 7065 samples (Appendix I). With this information, the experiment designs an appropriate number of additional trials for each test, denoted in Table 4.

Table 4. Repeated measures to meet minimum 7065 power sample size

Test #	EPS Level	Trials	Total Samples
5,12,19,26	10	707	7070
6,13,20,27	100	71	7100
1,2,3,4	500	15	7500
7,14,21,28	1000	8	8000
8,15,22,29	2500	3	7500
9,16,23,30	5000	2	10000
10,17,24,31	7500	1	7500
11,18,25,32	10000	1	10000

4.6.5 Timing.

It is important to have accurate and high-resolution timing measurements in this experiment because Response Time (RT) is the primary response variable. Therefore, Figure 19 illustrates the timestamping workflow developed to record the response time for each new log. The `QueryPerformanceCounter` function is a kernel-level operation that records a high resolution ($< 1\ \mu s$) timestamp [54] – these can then be used for time-interval measurements as the alert processes through the predefined AHNSR workflow. Lastly, the SDN controller is responsible for recording the end timestamp. This is accomplished using the native Linux command `# date +%s.%N`, which records the number of seconds since the UNIX epoch began, along with the nanosecond portion of the current system time.



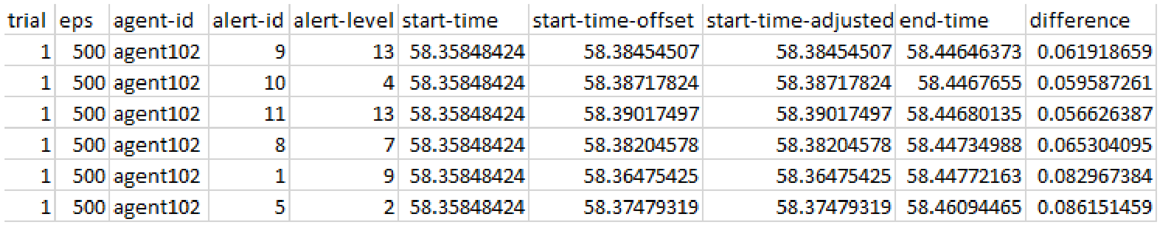
1) Event Occurs. Log is generated on each agent with timestamp
2) Client-side OSSEC *logcollector* process picks up log entry
3) Client's *agentd* process forwards log to OSSEC server's *remoted* process
4) OSSEC decoding and analysis trigger active-response script
5) OSSEC sends network modification request to Floodlight
6) Floodlight REST server accepts request, module handles it
7) Floodlight sends appropriate OpenFlow FLOWMOD message to switch
8) Switch updates its flow table(s)
9) Floodlight sends success message back to OSSEC server
10) Success message received. Timestamp logged on completing the network configuration

Figure 19. Log and alert timestamping through the AHSNR system

Additionally, VMtools has an option for virtual machines to sync with the ESXi server they are running on. The ESXi server includes Network Time Protocol (NTP) software and starts the *ntpd* service by default [55]. With all VM hosts connecting to the ESXi NTP server, system time synchronization can occur. Both the beginning and end timestamps can be compared against the MySQL timestamp artifact in the "middle" of the process, which guarantees the validity of the other timestamps.

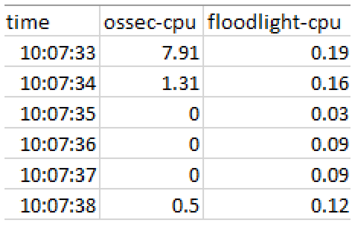
4.7 Statistical Analysis

A sample experiment, with n EPS, produces two data files for analysis: one RT-data.csv and one CPU-data.csv. Figures 20 and 21 display a sample portion of this data from a preliminary experiment with 500 EPS. All information necessary for metric parsing and analysis is recorded in these files (such as timestamps, trial numbers, `mpstat` output, etc.). Additional information, such as the individual alert id and level (per agent), is also captured during database retrieval for potential trend analysis. The maximum row index value also accounts for the AG metric, and should be equal to $eps \times number\ of\ trials$ for a 100% success rate in a single experiment.



trial	eps	agent-id	alert-id	alert-level	start-time	start-time-offset	start-time-adjusted	end-time	difference
1	500	agent102	9	13	58.35848424	58.38454507	58.38454507	58.44646373	0.061918659
1	500	agent102	10	4	58.35848424	58.38717824	58.38717824	58.4467655	0.059587261
1	500	agent102	11	13	58.35848424	58.39017497	58.39017497	58.44680135	0.056626387
1	500	agent102	8	7	58.35848424	58.38204578	58.38204578	58.44734988	0.065304095
1	500	agent102	1	9	58.35848424	58.36475425	58.36475425	58.44772163	0.082967384
1	500	agent102	5	2	58.35848424	58.37479319	58.37479319	58.46094465	0.086151459

Figure 20. Sample RT-data.csv file



time	ossec-cpu	floodlight-cpu
10:07:33	7.91	0.19
10:07:34	1.31	0.16
10:07:35	0	0.03
10:07:36	0	0.09
10:07:37	0	0.09
10:07:38	0.5	0.12

Figure 21. Sample CPU-data.csv file

After collecting data, a statistical analysis is performed using R, a GNU project language for statistical computing. First, the Response Time (RT) data are tested for mean validity using a one-sample t-test, and computing the standard deviation, mean, and 95% confidence interval.

The comparisons between the baseline control (Events Per Second = 0) and a particular treatment consists of paired t-tests or Welch-based t-tests. If pilot tests produce significant degrees of freedom above the necessary statistical power, then the paired t-tests are more appropriate. However, if additional runs are required for some but not all treatments, then the corresponding analysis uses the Welch t-test, which is more appropriate when sample sizes are very different.

A "full" general linear model is developed using all factors, and analyzed using R to identify significant and insignificant factors. This leads to a reduced model, which can be compared against the full model using Analysis Of Variance (ANOVA) techniques.

Due to performance metrics that may be related (e.g., CPU utilization increasing may affect overall network latency), an analysis of covariance general linear model is carried out against the experiment data.

4.8 Methodology Summary

This chapter describes the experimentation methodology used to measure the efficiency (CPU utilization) and effectiveness (Response Time and Alerts Generated) of the AHNSR system. The treatments use varying workload levels to determine the operational capacity of the system.

Experimental pilot tests reveal an average of 0.63675 seconds from log generation to flow table update. The test environment follows the configuration described, including two hosts, one OSSEC server, one Floodlight SDN controller, and one PICA P3290 switch.

V. Results and Analysis

5.1 Overview

This section describes the results obtained using the AHNSR design during the experiment structure described in Chapter 4. Section 5.2 describes initial observations of log collection and gives interpretative insight of the behavior. Analysis of Tests 1–4 are reported in Section 5.3, while the extra load testing (Tests 5–32) is reported in Section 5.4. Lastly, Section 5.5 describes how R is used to develop performance models of Response Time (RT) that fit the collected data. Interpretative discussion includes examples of how these models give insight into any given instance of the AHNSR framework.

5.2 OSSEC Log Collection Behavior

While conducting initial tests of OSSEC performance and behavior, the resulting response times were not following expected trends. As OSSEC advertises real-time monitoring of any host log files, this research expected each log entry to be forwarded immediately to the OSSEC server and continue down the OSSEC log pipeline described in Section 3.4. Therefore, each individual log event would result in very similar, or near constant response time under normal load.

However, preliminary tests reveal something different. Figure 22 shows the individual alert response times of 250 log entries entered into a single host's syslog file, following the experimental design setup described in Chapter 3. The scatter-plot displays non-constant response times, and also presents a significant "jump" around alert ID 160.

After scaling the tests to multiple hosts, this pattern continues. Figure 23 displays a similar preliminary test on 10 different hosts simultaneously (Agent IDs Agent 101

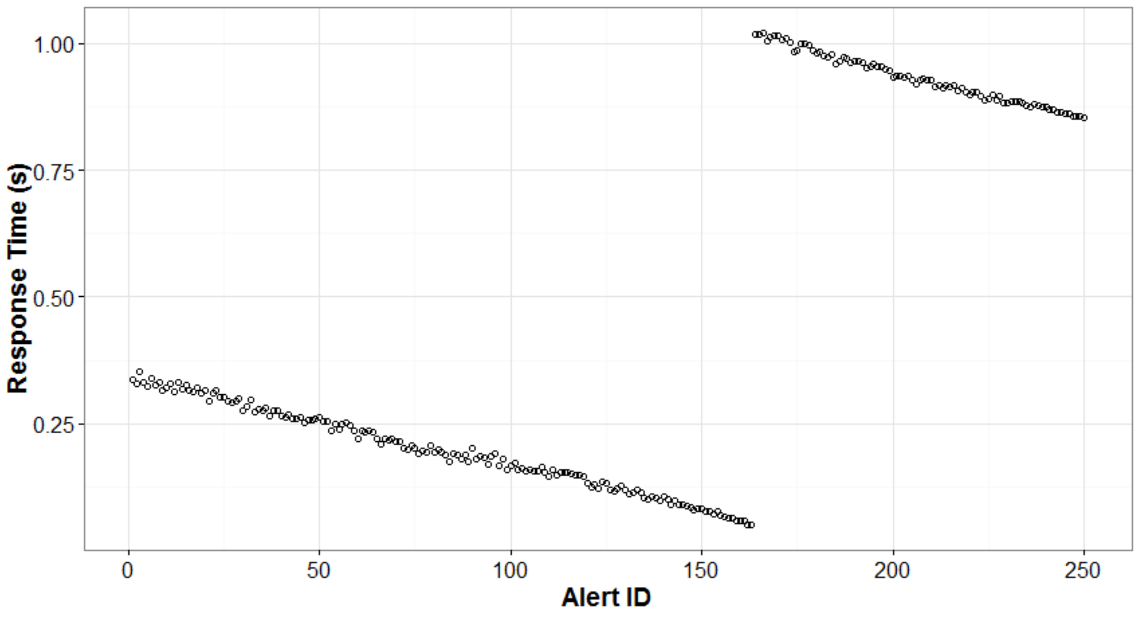


Figure 22. Individual alert response time from 250 logs collected on one host

to Agent 110). By following the data-points of each agent's logs, the same pattern emerges: sequentially decreasing times, followed by an approximate 1 second sudden response time increase.

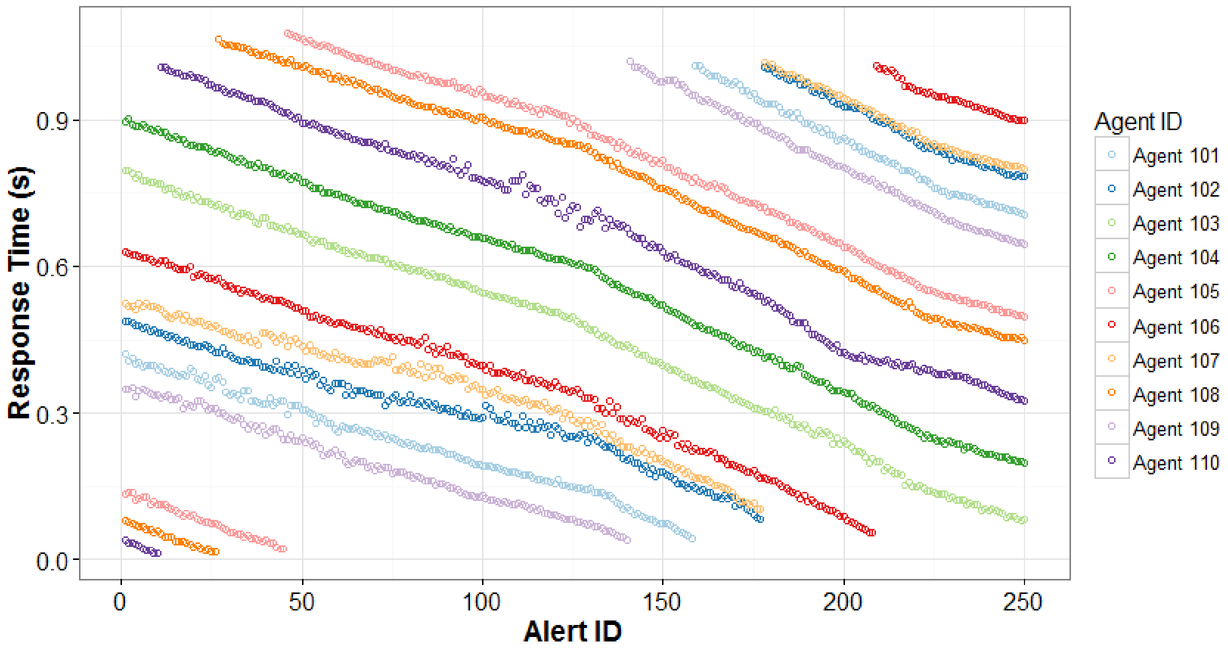


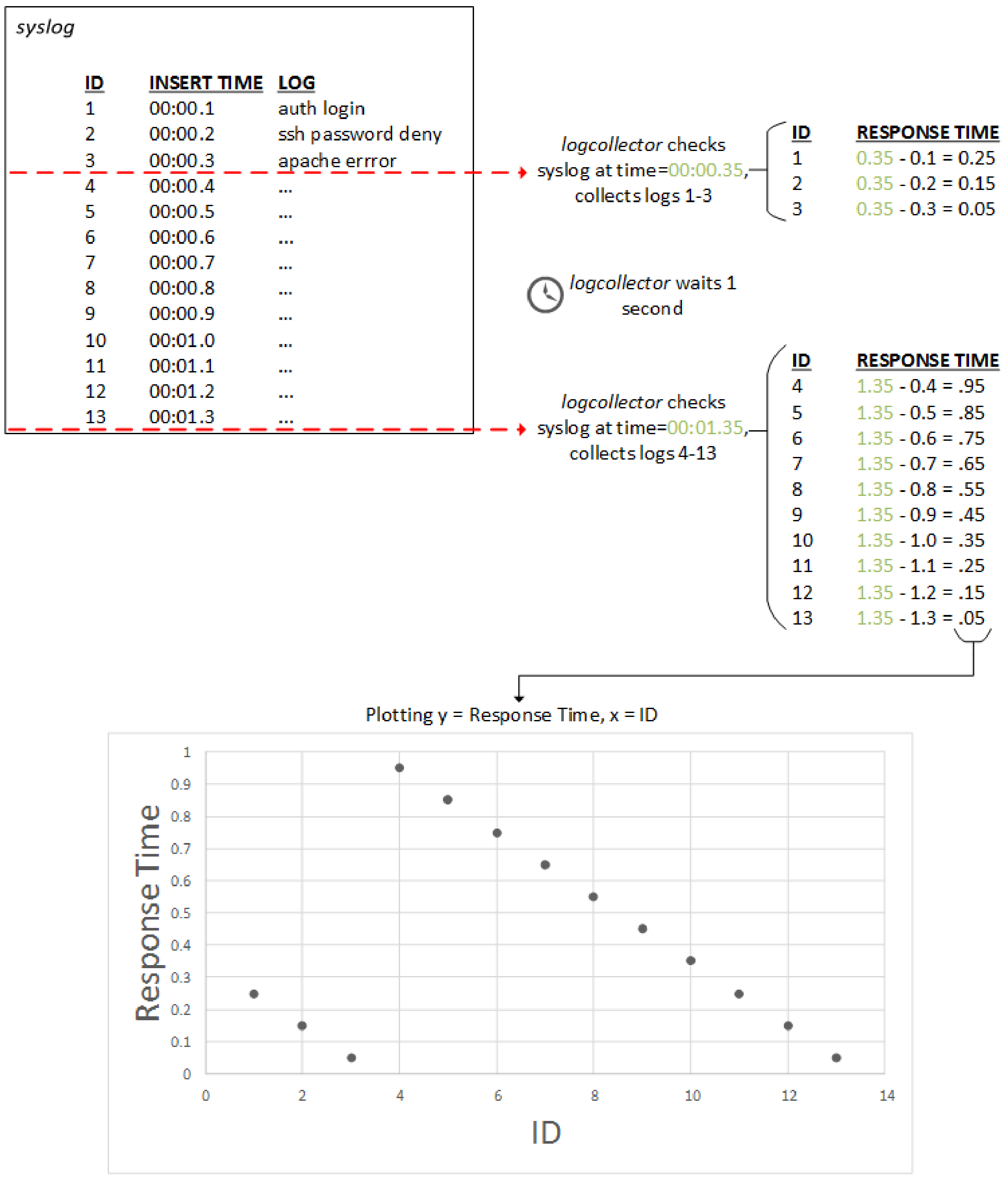
Figure 23. Individual alert response time from 250 logs per host, from 10 hosts

Upon further investigation, the source code of *logcollector.c*, a global variable exposes the truth that its daemon is not actually forwarding the logs real-time, but is instead executing inside a while loop defined by a *loop_timeout* variable. Furthermore, an internal OSSEC options configuration file has a *logcollector.loop_timeout* property set to one, its minimum accepted value (Appendix D). Documentation states the property units are in seconds – this suggests any new logs are collected after every 1 second timeout, which can help explain the trends in Figure 22.

Figure 24 helps illustrate the entire process in detail. A sample experiment begins logging entries from a relative time point of zero. The agent's *logcollector* daemon has been running continually, and its current timeout happens to expire at time=0.35s, at which point it immediately checks the monitored syslog file, sees the 3 new log entries, and forwards them to the server for processing. In calculating the response time for these 3 logs, the difference between the singular check time (0.35) and the naturally increasing insertion times results in decreasing values (.25 $\rightarrow$ .15 $\rightarrow$ .05). After collecting those logs, the daemon re-enters a one second timeout, checking the log again at time=1.35s. At this point, it collects the new entries 4-13, again forwarding them for processing. Similarly, the calculated response times result in a sequentially decreasing trend, and with all the response times plotted against their Alert ID, the graph in Figure 24 mirrors the actual results seen in Figures 22 and 23.

The slight differences in sequential alerts is the sum of both write and read operations: the actual write time of logs to the syslog file, in addition to the first-in, first-out (FIFO) processing times during the checks. This gives the most advantageous response times to those logs inserted most recent to the point of log collection.

Preliminary results in Figure 23 also show how the agents' *logcollector* daemons are not synchronized together, as the offsets of each 1 second timeout result in offsets of the recorded response times. Their timeout sessions operate independently, yet given



syslog

ID	INSERT TIME	LOG
1	00:00.1	auth login
2	00:00.2	ssh password deny
3	00:00.3	apache errror
4	00:00.4	...
5	00:00.5	...
6	00:00.6	...
7	00:00.7	...
8	00:00.8	...
9	00:00.9	...
10	00:01.0	...
11	00:01.1	...
12	00:01.2	...
13	00:01.3	...

logcollector checks syslog at time=00:00.35, collects logs 1-3

ID	RESPONSE TIME
1	0.35 - 0.1 = 0.25
2	0.35 - 0.2 = 0.15
3	0.35 - 0.3 = 0.05

logcollector waits 1 second

logcollector checks syslog at time=00:01.35, collects logs 4-13

ID	RESPONSE TIME
4	1.35 - 0.4 = .95
5	1.35 - 0.5 = .85
6	1.35 - 0.6 = .75
7	1.35 - 0.7 = .65
8	1.35 - 0.8 = .55
9	1.35 - 0.9 = .45
10	1.35 - 1.0 = .35
11	1.35 - 1.1 = .25
12	1.35 - 1.2 = .15
13	1.35 - 1.3 = .05

Figure 24. Sample log collection behavior with a 1 second timeout loop

enough data (discussed in Section 4.7), statistically significant conclusions can still be made about the system behavior. After understanding the log collection behavior, it is apparent the recorded response time performance should still converge on a statistical mean with $<1s$ confidence interval, despite the implicit range of varying timeout offsets.

5.3 Performance Under Expected Load

This section analyzes AHNSR performance results under the suggested normal operating load of 500 EPS (discussed in Section 4.6).

Table 5 summarizes the results of Tests 1–4, which individually test the SDN Response factors at the given operational load level. All responses provide 100% successful alert generation. Both CPU utilization and Response Time vary among the tests – the proceeding analysis determines the significance of each.

Table 5. AHNSR performance results under 500 EPS load, 10 active agents

Test #	**Response**	**AG**$_{(\%)}$	**CPU**$_{(\%)}$	**Mean RT**$_{(s)}$	σ	$\sigma_{\bar{x}}$	**95% C.I.**$_{(s)}$
(control)	N/A	N/A	0.20	N/A	N/A	N/A	N/A
1	Log-Only	100	9.22	0.5210	0.2860	0.0040	±0.0079
2	Static Flow	100	34.45	0.5639	0.2779	0.0039	±0.0077
3	ACL	100	28.52	0.5438	0.2825	0.0040	±0.0078
4	Firewall	100	31.88	0.5711	0.2934	0.0041	±0.0081

5.3.1 Expected Load: Response Time.

Figure 25 displays the spread of all RT data for Tests 1–4 (individual trials shown in Appendix K). The ranges correlate with the behavior expected in the log collection workflow; however slight differences are visible, especially with the dispersion of the upper quartile in the Firewall test.

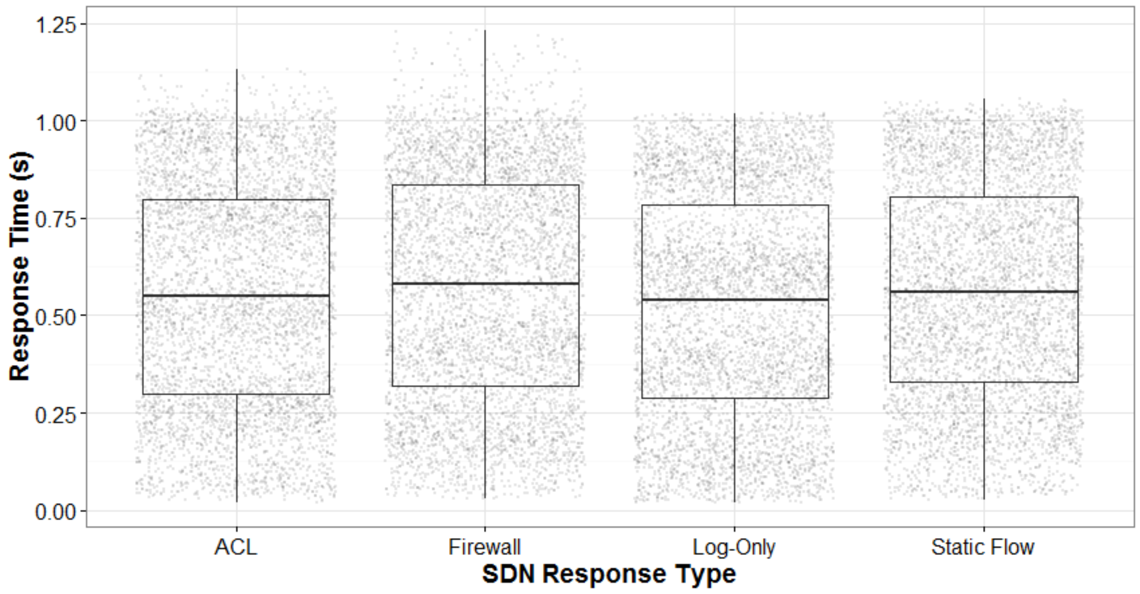


Figure 25. RT quartile ranges from 15 trials, 500 EPS, 10 active agents

The R script in Appendix I produces Figure 26, which is a plot of resultant confidence intervals on the means based on the calculations of standard deviation, standard error of the mean, and a confidence interval (default 95%). The analysis continues by conducting other statistical methods for further insight.

A one-way repeated measures ANalysis Of VAriance (ANOVA) test provides an F-statistic of 30.189 and a p-value $< 2.2e^{-16}$. Therefore, this research clearly rejects the null hypothesis of equal means for all four response types at the 500 EPS event density. The ANOVA test only answers the question of whether or not there are significant differences in the response time means. However, it does not provide us with any information about *how* they differ. The analysis must conduct a pairwise t-test to collect this information.

The pairwise t-tests use the Holm adjustment method to correct the Type I error rate across the tests. Mathematically, the Holm adjustment sequentially compares

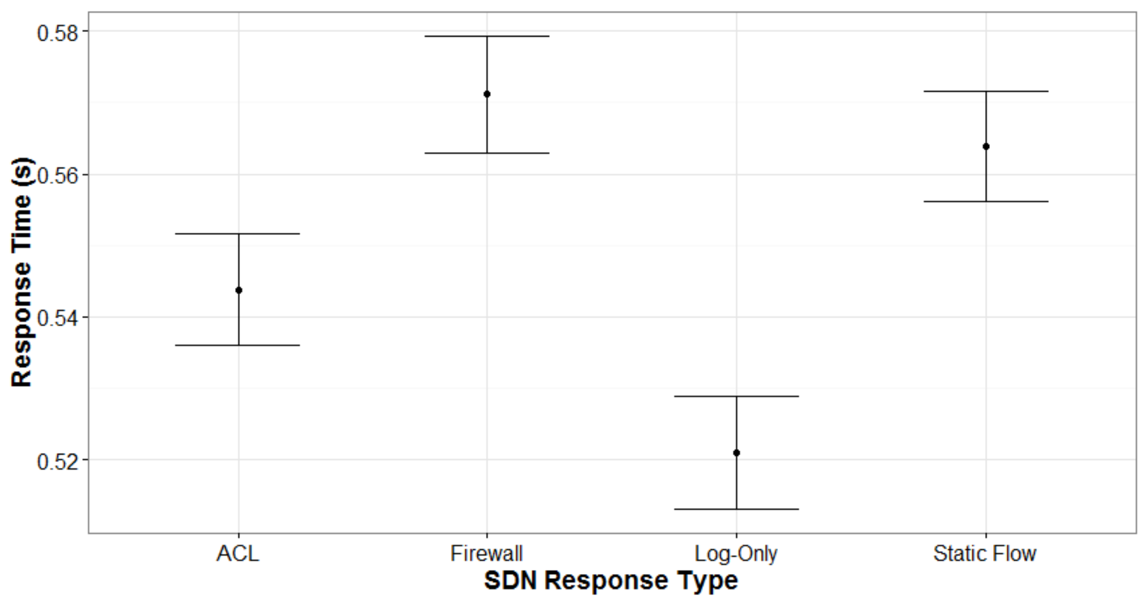


Figure 26. C.I. on the RT mean from 15 trials of 500 EPS, 10 active agents

the lowest p-value with a Type I error rate that is reduced for each consecutive test. The Holm method is generally considered superior to the Bonferroni adjustment, yet still maintains a mathematically conservative approach [56]. The test outputs the following:

	ACL	Firewall	Log-Only
Firewall	6.8×10^{-6}	-	-
Log-Only	0.00019	$< 2 \times 10^{-16}$	-
Static Flow	0.00084	0.20737	2.6×10^{-13}

Using an alpha level of .05 threshold, the pairwise t-test results indicate that there is a statistically significant difference between almost all responses, excluding that between static flow and firewall ($p = .20737 > \alpha$). Therefore, the analysis of Tests 1–4 can develop the following statistical conclusions:

1. At this load level (500 EPS), the Log response is faster than all other tested responses.

2. At this load level (500 EPS), the ACL response is faster than the Firewall and Static Flow response.

5.3.2 Expected Load: CPU.

CPU data is measured using the Linux `mpstat` package, and Figure 27 displays the spread of CPU data for each SDN response at the tested level, 500 EPS. The data only includes a single measurement captured during the first second after the experiment begins, because RT measurements show the active response completing within one second on average. Therefore the tests only capture CPU utilization directly involved in active response pre-processing. Any measurements below or equal to the control threshold ($\leq 0.20\%$ – from "control" test in Table 5) would represent behavior equivalent to the system at an idle state (reduction seen in Appendix L). The repeated measures design provides additional data for the CPU metric by having multiple trials with every SDN response type, as discussed in Section 4.6.4.

While there are technically two servers with CPU resources (OSSEC and Floodlight), only the mean CPU utilization of the OSSEC server is reported on Table 5 because it is the primary consumer. Both servers' data are visible in Figure 27. The ranges for the CPU metric on the OSSEC server are higher in comparison to the counterpart Floodlight utilization during and given SDN response type test. It follows logically that the utilization range on the Floodlight server during the Log-Only test results in a narrow range near zero. The Log-Only response does not demand any immediate work from the specialized modules provided by Floodlight, so one would expect to see a minimal amount of processor cycles used.

Likewise with the RT data, the analysis proceeds to extrapolate the standard deviation and standard error of the measurements in order to build a 95% confidence interval on the mean utilization for each server-response pair. Figure 28 displays these results vertically stacked on each SDN response.

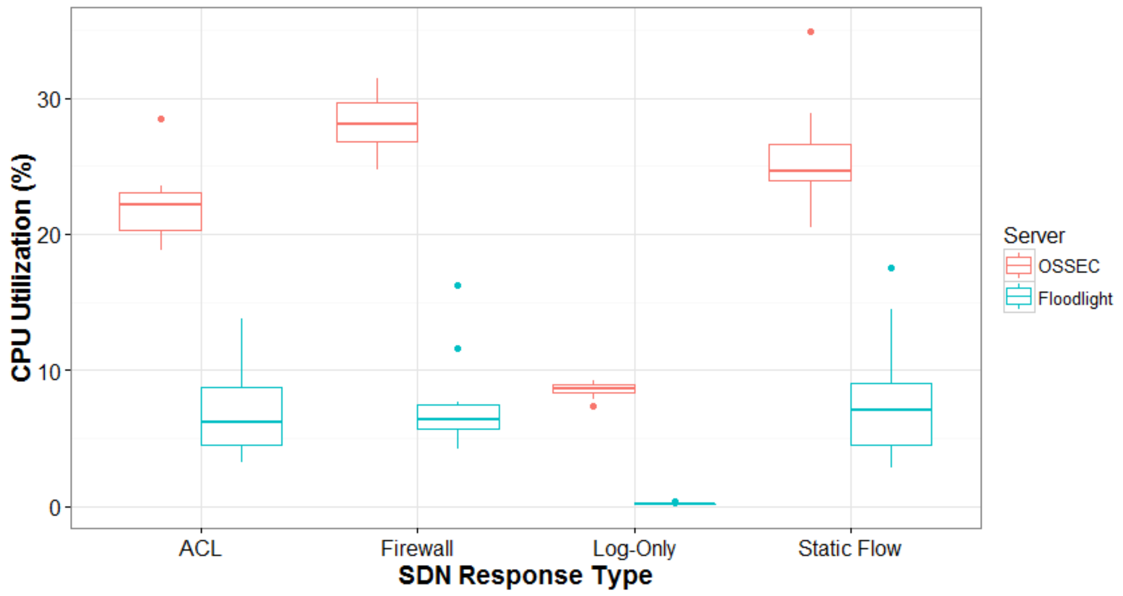


Figure 27. OSSEC and Floodlight CPU quartile ranges, 15 trials at 500 EPS

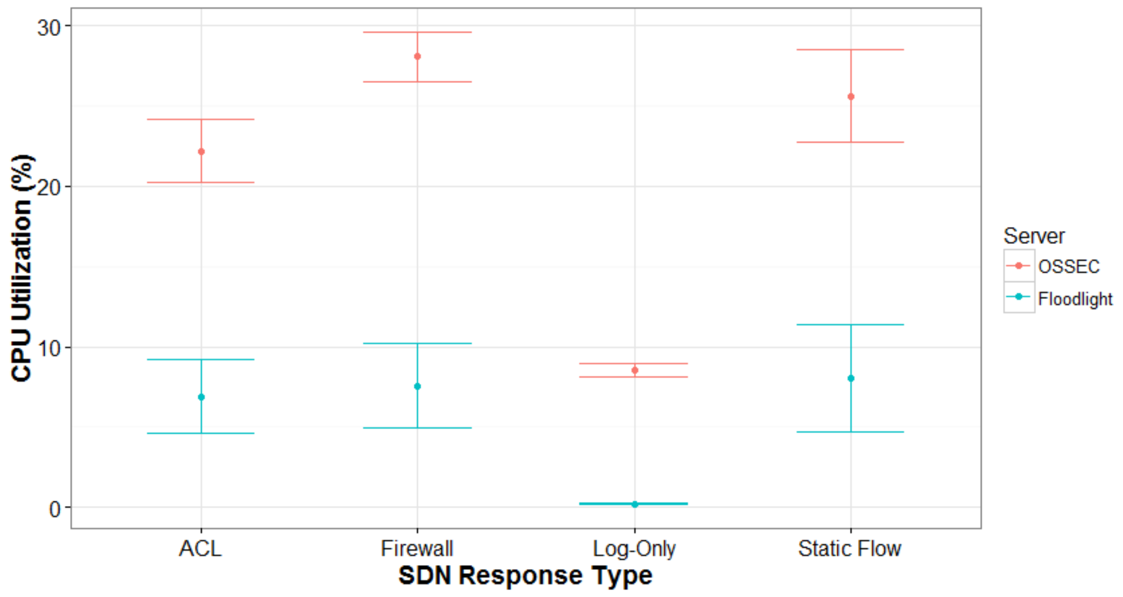


Figure 28. C.I. on the Floodlight and OSSEC CPU mean, 15 trials of 500 EPS

An ANOVA test on each set of the CPU data from OSSEC and Floodlight produces F statistics of 105.98 and 11.7373 respectively. This guarantees that the means are not all equal across the SDN type subsets. Furthermore, a Holm-adjusted pairwise t-test for the CPU utilization between the responses on the OSSEC server outputs the following:

	ACL	Firewall	Log-Only
Firewall	6.1×10^{-5}	-	-
Log-Only	6.6×10^{-13}	$< 2 \times 10^{-16}$	-
Static Flow	0.014	0.049	1.1×10^{-15}

Similarly, a Holm-adjusted pairwise t-test for the CPU utilization between the responses on the Floodlight server outputs the following:

	ACL	Firewall	Log-Only
Firewall	1.0	-	-
Log-Only	0.00036	0.00012	-
Static Flow	1.0	1.0	5.6×10^{-5}

Using the same alpha threshold of .05, the analysis can come to the following statistical conclusions:

1. There are significant differences in the CPU means between all SDN response methods on the OSSEC server.

2. There is only a significant difference in the CPU mean for the Log-Only response method on the Floodlight server.

This research considers the net sum of all resources from both servers when determining the most efficient response method. Therefore, the evaluation of Tests 1–4 orders the response methods from best to worst in terms of CPU resource efficiency: Log-Only, ACL, Static Flow, and Firewall.

5.4 Load Testing

In order to gather insight on AHNSR performance under non-standard load levels (e.g., peak periods), Tests 5–32 measure how it responds under increasing event density levels. The summary of these results are shown in Tables 6 through 9. General observations from the summary data can be formulated:

- Every EPS level resulted in 100% successful alert generation.
- There is a trend of increasing CPU as EPS increases.
- There is a trend of increasing RT as EPS increases.

Each EPS level is equivalent to the aggregate log amount generated from all agents. For example, the 100 EPS level is the result of a 10 logs entered on 10 disparate agents. With this design, and considering the growing sample size for subsequent tests, the tests set a sample size baseline equal to the power test calculated in Section 4.6.4 ($n \geq 7065$). Therefore, all EPS levels except 7500 and 10000 require repeated trials in a repeated measures design to meet the minimum sample size for analysis.

Table 6. Log-Only response load testing results

Test #	**EPS**	**AG**$_{(\%)}$	**CPU**$_{(\%)}$	**Mean RT**$_{(s)}$	σ	$\sigma_{\bar{x}}$	**95% C.I.**$_{(s)}$
(control)	0	N/A	0.20	N/A	N/A	N/A	N/A
5	10	100	0.31	0.4941	0.2935	0.02075	±0.0409
6	100	100	2.06	0.4812	0.2867	0.0064	±0.0274
7	1000	100	13.4	0.5134	0.2640	0.0026	±0.0176
8	2500	100	27.99	0.5415	0.2883	0.0057	±0.0113
9	5000	100	60.15	0.7623	0.3259	0.0046	±0.0090
10	7500	100	60.84	1.0396	0.4334	0.0050	±0.0098
11	10000	100	61.06	1.1514	0.4892	0.0048	±0.0095

Table 7. ACL response load testing results

Test #	**EPS**	**AG**$_{(\%)}$	**CPU**$_{(\%)}$	**Mean RT**$_{(s)}$	σ	$\sigma_{\bar{x}}$	**95% C.I.**$_{(s)}$
(control)	0	N/A	0.20	N/A	N/A	N/A	N/A
12	10	100	0.56	0.5174	0.2872	0.0064	±0.0125
13	100	100	3.83	0.5599	0.2872	0.0064	±0.0253
14	1000	100	44.34	0.7050	0.3131	0.0031	±0.0221
15	2500	100	76.49	1.1243	0.4027	0.0080	±0.0157
16	5000	100	80.06	2.0061	0.7462	0.0105	±0.0206
17	7500	100	78.63	2.4270	0.9210	0.0106	±0.0208
18	10000	100	79.62	3.2080	1.1990	0.0119	±0.0235

Table 8. Firewall response load testing results

Test #	EPS	AG$_{(\%)}$	CPU$_{(\%)}$	Mean RT$_{(s)}$	σ	$\sigma_{\bar{x}}$	95% C.I.$_{(s)}$
(control)	0	N/A	0.20	N/A	N/A	N/A	N/A
19	10	100	0.72	0.5206	0.2800	0.0197	±0.0390
20	100	100	4.51	0.5209	0.2942	0.0065	±0.0261
21	1000	100	55.73	0.9120	0.3394	0.0107	±0.0113
22	2500	100	80.94	1.3112	0.6265	0.0056	±0.0210
23	5000	100	85.49	2.8844	1.1538	0.0163	±0.0319
24	7500	100	87.13	5.2275	2.1906	0.0254	±0.0499
25	10000	100	87.39	5.8603	2.2177	0.0222	±0.0435

Table 9. Static Flow response load testing results

Test #	EPS	AG$_{(\%)}$	CPU$_{(\%)}$	Mean RT$_{(s)}$	σ	$\sigma_{\bar{x}}$	95% C.I.$_{(s)}$
(control)	0	N/A	0.20	N/A	N/A	N/A	N/A
26	10	100	0.66	0.5340	0.2912	0.0291	±0.0577
27	100	100	4.97	0.5204	0.2868	0.0064	±0.0175
28	1000	100	53.04	0.5752	0.3573	0.0035	±0.0172
29	2500	100	69.7	1.3531	0.5025	0.0100	±0.0249
30	5000	100	75.46	2.4499	1.0372	0.0146	±0.0287
31	7500	100	75.76	2.7983	1.1034	0.0127	±0.0249
32	10000	100	76.44	3.7025	1.4978	0.0149	±0.0293

5.4.1 AG.

To test these higher-end networking load levels on the servers, modifications to the Linux kernel networking parameters were necessary. Appendix H details the tuning changes made on both the OSSEC and Floodlight servers. These modifications included an internal change to the system file descriptor limit because every open network socket requires a file descriptor in Linux. Increasing this limit ensures that lingering TIME_WAIT sockets and other consumers of file descriptors do not impact the ability to handle numerous concurrent requests. Therefore, this limit must be set above the EPS load level in each test environment because each event subsequently initiates a TCP connection between OSSEC and Floodlight, creating open network socket file descriptors.

The results show AHNSR successfully handles every EPS load level with 100% alert generation. Interestingly, OSSEC's agent to server communication dependency on UDP (an implicit potential weakness of OSSEC's log collection procedure) did not seem to affect its ability to handle even extreme loads. Granted, the lack of normal enterprise IP traffic may affect this ability, but it should still maintain at least a normally observed standard of 98% reliable delivery [57]. Additionally, a single intrusion would commonly result in several logs (indicative of the threat) forwarded to the OSSEC server – thus, it could be argued that OSSEC can afford to lose a certain percentage of those logs if at least one results in the appropriate active response action.

5.4.2 Load Testing: RT.

Comparing the difference in adjusted start time and response end time for each alert in the data, this section summarizes and analyze the RT metric from both a global and per-test perspective. For each EPS level, the mean RT is calculated along with its standard deviation, standard error, and resulting 95% confidence interval (R code in Appendix J).

Figure 29 displays these means for each SDN response at the tested EPS levels. A small colored ribbon connects the high and low confidence intervals between points. Interestingly, the seemingly minor, insignificant differences seen at the 500 EPS level in Tests 1–4 are definitely magnified at increasing loads. For example, the widest observed gap is 4.67 seconds, occurring between the upper C.I. on the Log-Only mean RT and the lower C.I. on the Firewall mean RT at the 10000 EPS level. Given these results, administrators should consider the significant performance hit when using the Firewall module as an active response, especially if similar results can be obtained by utilizing the Static Entry or ACL modules.

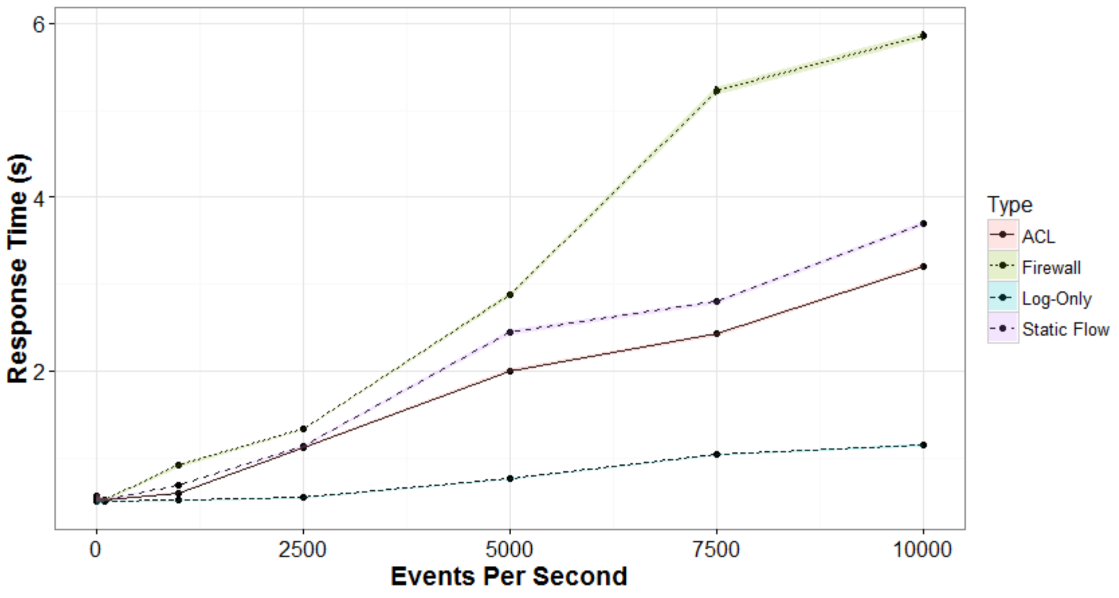


Figure 29. Mean RT for EPS levels 10 through 10000, 10 active agents

Furthermore, these results can give insight into an appropriate system architecture based on network needs and load estimation. If a network with similar resources to this experiment anticipates peak periods of 5000 SDN Response Demanding EPS, and certain policies state that any automated immediate reaction to any intrusion must be, on average, under 2 seconds – these results suggest the limitations would eliminate the Firewall and Static Flow Entry active responses as acceptable candidates.

5.4.3 Load Testing: CPU.

This section reports and analyzes the average utilization of the AHNSR system's CPU resources during load testing. The sample size of this data is much smaller than that of the RT data, due to interval limitations of `mpstat` output (as noted in Section 1.6). However, the analyis can still come to conclusions about the effect of EPS on this metric.

Since some RT measurements are greater than one second, a more accurate picture of resource utilization includes the entirety of the `mpstat` output, which collects CPU information for 10 seconds, when AHNSR work may still be occuring. Therefore, Figure 30 plots the mean of the CPU measurements for each SDN response at every EPS level. It follows the order of performance that has been observed in previous tests, with the Log-Only response maintaining the lowest utilization, Firewall maintaining highest, and ACL/Static Flow in between.

The collected data seems to follow a logarithmic trend (with the exception of Log-Only response) that tends to level off at an upper limit for each response. One explanation for this could be the Linux operating system's CPU scheduler trying to meet the needs of both bash execution and network I/O. Resource limits and cycling between process and I/O bursts can extend the CPU utilization required over multiple seconds versus what is possible in a single second. Also, testing of the log generation script suggests that the experiments are reaching the upper limit of how many logs can be written to the syslog file per second. This would correlate to utilization *per second* leveling off as the actual EPS generation rate *per second* is the same. Once pass the maximum threshold, it would require multiple seconds to actually generate

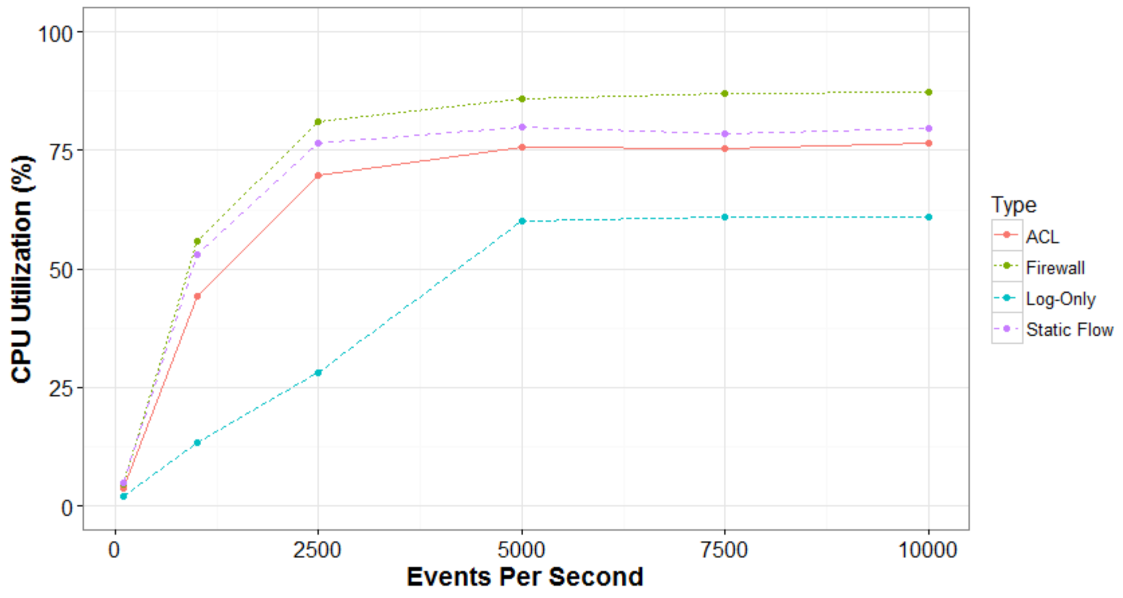
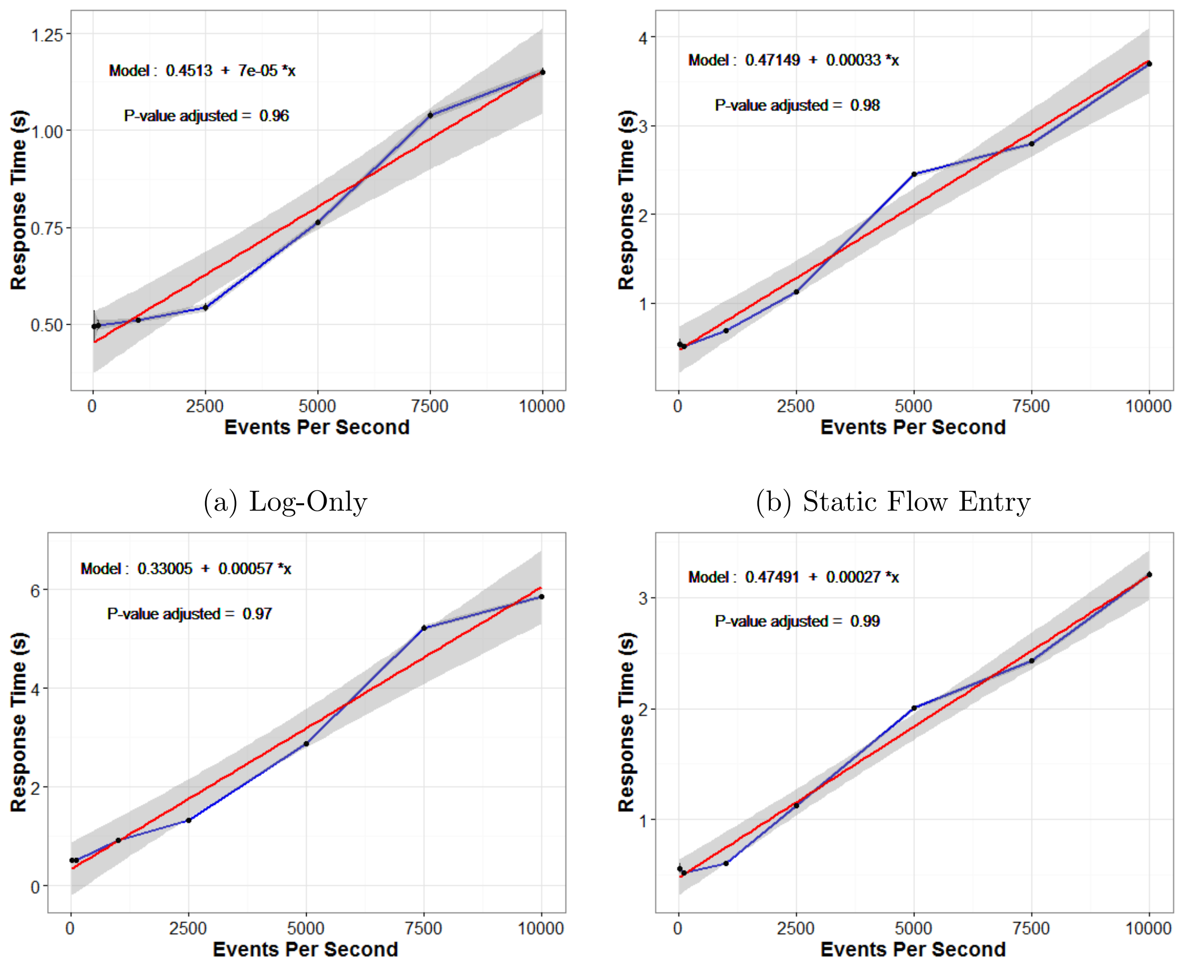


Figure 30. Mean CPU utilization for EPS levels 10 through 10000, 10 active agents

the amount of logs. Further testing confirms this – the log generation script has a max generation rate near 1000 logs per second. This means testing an aggregate EPS level above 10000 would require more than ten agents ($1000 \times 10 = 10000$).

5.5 RT Performance Models

Using the collected data, R constructs models of AHNSR system performance. Appendix J details the necessary R code execution in developing a linear model for each SDN response. Considering the primary response variable, RT, as the output, Figure 31 plots in blue the mean RT summary data from Tables 6–9. The red line illustrates the corresponding linear models for each response with a surrounding 95% confidence interval.



(a) Log-Only (b) Static Flow Entry

(c) Firewall (d) ACL

Figure 31. Fitting linear models where y = Response Time and x = EPS

These models represent systems that choose to implement that specific SDN response style exclusively. When RT (Response Time) is described as a function of eps (Events Per Second), the models can be derived as

- **Log-Only:**

$$RT = 0.42869 + 0.00007eps \quad (3)$$
$$RT_{[95\%\ lower]} = 0.33000 + 0.00005eps \quad (4)$$
$$RT_{[95\%\ upper]} = 0.52737 + 0.00009eps \quad (5)$$

- **Static Flow Entry:**

$$RT = 0.46778 + 0.00033eps \quad (6)$$
$$RT_{[95\%\ lower]} = 0.07721 + 0.00024eps \quad (7)$$
$$RT_{[95\%\ upper]} = 0.85834 + 0.00040eps \quad (8)$$

- **Firewall:**

$$RT = 0.23732 + 0.00059eps \quad (9)$$
$$RT_{[95\%\ lower]} = -0.50995 + 0.00045eps \quad (10)$$
$$RT_{[95\%\ upper]} = 0.98458 + 0.00072eps \quad (11)$$

- **ACL:**

$$RT = 0.49669 + 0.00027eps \quad (12)$$
$$RT_{[95\%\ lower]} = 0.31284 + 0.00025eps \quad (13)$$
$$RT_{[95\%\ upper]} = 0.68054 + 0.00030eps \quad (14)$$

5.5.1 Dynamic Model.

If the AHNSR framework were to use a combination of different responses, R must construct a model to include all parameters. This is a more flexible model that allows for a dynamic response policy in real enterprise environments. However, in order to correctly analyze the data in this fashion, this research cannot interpret the parameters as binary factors. Instead, they can be considered as continuous, numeric values representing the percentage of alerts that will be attributed to its response method. Therefore, the current analysis interprets the conducted tests as

Table 10 suggests, where each test group is actually the sum of the percentages for each response method and that specific category maintains 100% alert attribution.

Table 10. Dynamic model interpretation

Test #	EPS	Alert Distribution To			
		Log-Only	Firewall	ACL	Static
5-11	10, 100, 1000, 2500, 5000, 7500, 10000	100%	0%	0%	0%
12-18	10, 100, 1000, 2500, 5000, 7500, 10000	0%	100%	0%	0%
19-25	10, 100, 1000, 2500, 5000, 7500, 10000	0%	0%	100%	0%
26-31	10, 100, 1000, 2500, 5000, 7500, 10000	0%	0%	0%	100%

Because each method is operating with independent active response scripts and Floodlight modules, this research assumes there will be no dependency issues affecting the performance metrics when starting to mix the methods in this dynamic model. This assumption allows the analysis to use the previously collected experimental data to develop a model and make educated estimates on the mixed scenario described next.

The derived expression for the average response time (RT_{avg}) in a dynamic model can be described as

$$RT_{avg} = l(0.42869 + 0.00007e) + s(0.46778 + 0.00033e)+ \quad (15)$$
$$f(0.23732 + 0.00059e) + a(0.49669 + 0.00027e)$$
$$RT_{avg[95\%\ lower]} = l(0.33000 + 0.00005e) + s(0.07721 + 0.00024e)+ \quad (16)$$
$$f(-0.50995 + 0.00045e) + a(0.31284 + 0.00025e)$$
$$RT_{avg[95\%\ upper]} = l(0.52737 + 0.00009e) + s(0.85834 + 0.00040e)+ \quad (17)$$
$$f(0.98458 + 0.00072e) + a(0.68054 + 0.00030e)$$

where RT_{avg} is described as a function of e (EPS), a (ACL response ratio), f (Firewall response ratio), l (Log-only response ratio), and s (Static Flow response ratio). This function assumes $(a + f + l + s) = 1$, meaning the distribution ratios sum to 100%.

This consolidated dynamic model can be used where general network traffic statistics are already known. For example, the distribution ratios are populated by taking a sample network environment with the following estimates (these values are based upon the immediate active response required on the aggregate log amount in the AHNSR workflow – thus a brief rationale is included for the estimation in a generalized enterprise environment):

1. **80% Log-Only:** A traditional enterprise environment can safely assume that most logs need no further action other than OSSEC logging.

2. **10% ACL:** Some suspicious, threatening, or multi authentication-failed logs may require IP addresses to be put on the access control list temporarily or permanently.

3. **5% Firewall:** Since the firewall is configured relatively static for system control over ports/services/etc., only extreme cases should create new rules to imple-

ment enterprise-wide.

4. **5% Static Flow:** Special circumstances may require specific, custom flow responses that can be specific to IP, hostname, username, or other match fields.

With these estimates inserted into the dynamic RT function, the average RT becomes

$$\begin{aligned} RT_{avg} = {} & .8(.42869 + .00007e) + .1(.46778 + .00033e) + \\ & .05(.23732 + .00059e) + .05(.49669 + .00027e) \end{aligned} \tag{18}$$

Figure 32 is a plot of RT_{avg} for the consolidated dynamic model. Using this information, administrators can make informed decisions about network design and scalability requirements. For example, if they desire a near-immediate ($< 1s$) deny-access response when necessary, then the model suggests an appropriate EPS level. Given this limit, auditing policies on all workstations can be modified to produce an average aggregate less than 4000 logs per second (as Figure 32 shows the average RT near 4000 EPS $\leq 1s$). On the other hand, the threshold provides an estimate on the number of agents that a single AHNSR instance can support.

Lastly, this research seeks to confirm the accuracy of the dynamic model by conducting a single additional experiment with 10 agents each logging 500 EPS, for an aggregate 5000 EPS. However in this scenario, the active responses are configured to only respond for specific alert levels.

There are 18 possible alert levels in OSSEC and each log has a single alert level identifier, discussed in Section 3.4. Therefore, this test's distribution can mimic the sample network described above like so:

- Log-Only assigned to alert levels 1 through 14. [14/18 = 77.77% distribution]

- ACL assigned to alert levels 15 and 16. [2/18 = 11.11% distribution]

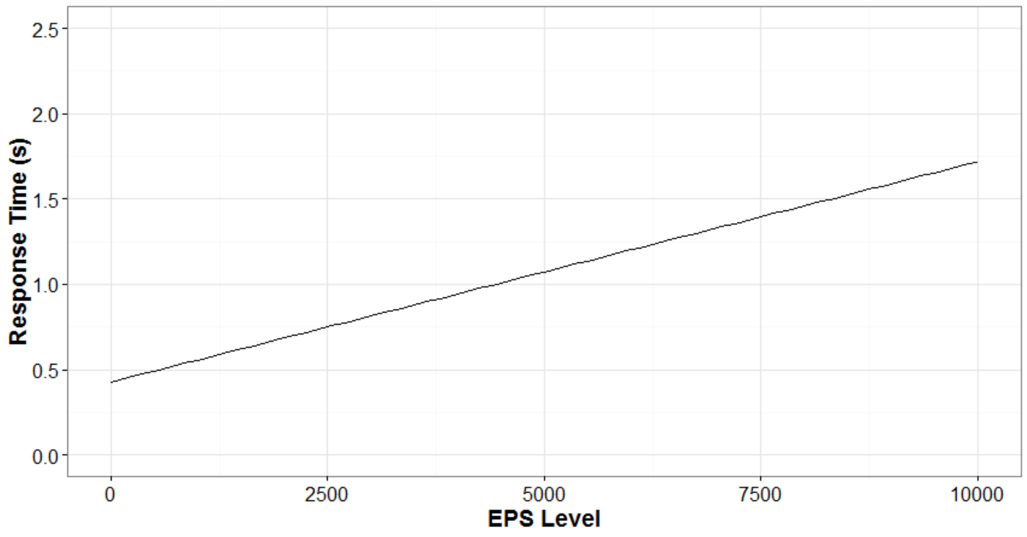


Figure 32. RT estimate in 80% Log, 10% ACL, 5% Firewall, 5% Static Flow model

- Firewall assigned to alert level 17. [1/18 = 5.55% distribution]
- Static Flow assigned to alert level 18. [1/18 = 5.55% distribution]

The test yields a mean RT of 1.049 seconds. Figure 33 shows how this data (represented by the blue X) falls within the 95% confidence interval for the dynamic model when the EPS Level = 5000, confirming the accuracy of the model.

5.6 Chapter Summary

This section summarizes the results of all evaluation tests. Preliminary results are discussed with insight on the log collection timeout behavior. Next, it presents data results of all tests, indicating when there were statistically significant differences in the RT and CPU performance metrics. The 100% success rate of the AG metric also indicates dependable use of the OSSEC log collection workflow. Lastly, it compares the performance between different active response methods and presents performance models for the RT factor.

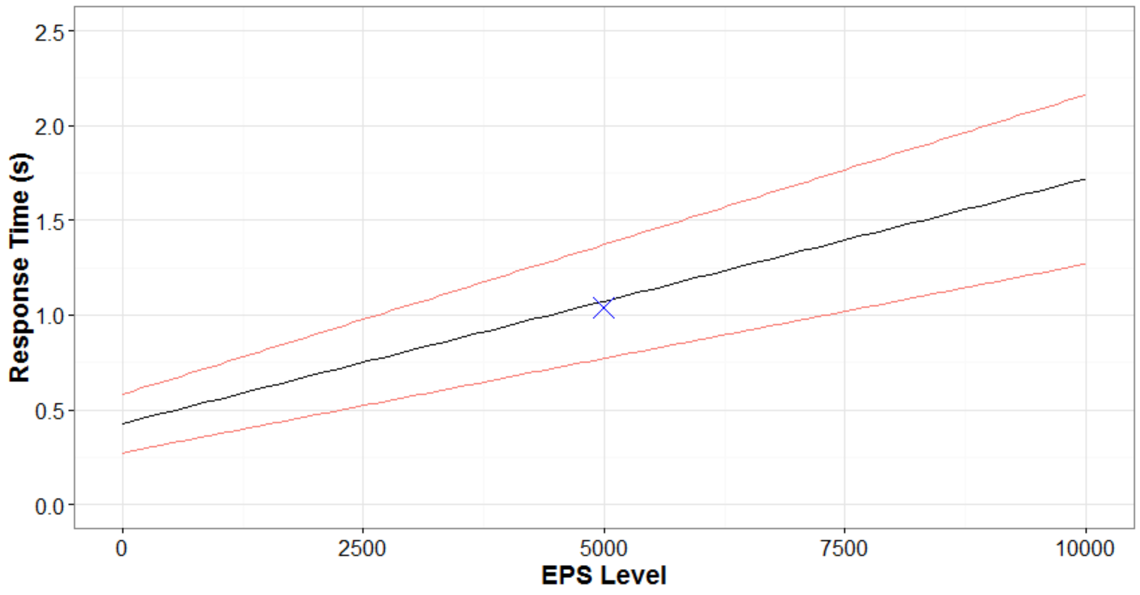


Figure 33. Dynamic model test result

VI. Conclusion

6.1 Overview

This chapter summarizes the research and statistical conclusions from the experimental evaluation. Section 6.2 reiterates key conclusions drawn from the research. Section 6.3 discusses experimental significance in the SDN domain. Finally, Section 6.4 suggests potential opportunities for future work with the AHNSR system design.

6.2 Research Conclusions

The research was successful in implementing a viable security solution for SDN controller action in response to host-based intrusion detection system alerts. As hypothesized, both the response time for successful completion of an appropriate SDN response and the processor resource utilization increased as the aggregate number of logs being generated by hosts (EPS) increased. The alert generation rate remained above an effectiveness standard of 99% for all tests. Additionally, when comparing the different response methods, the Access Control List response method did outperform the Firewall response method in both quantitative and qualitative metrics. These conclusions support the original hypothesis.

Evaluation tests emulate a traditional enterprise network using OSSEC server/agents deployment, real hardware, and implementation of security options. Security is not overlooked during AHNSR configuration as it adopts confidentiality, integrity, and authentication characteristics, building an appropriate defense against attacks directed towards its internal structure. All OpenFlow and REST API communication is encrypted via SSL/TLS, and pre-configured certificates allow whitelisting for higher privilege access of trusted systems.

Tests 1–4 demonstrate a mean active response time near 0.53 seconds, regardless of response method, when the system is handling 500 incoming logs per second. Also, CPU utilization on all components remained below 35%. These metrics illustrate an efficient and effective integration of OSSEC and Floodlight in the AHNSR design.

Load testing from Tests 5–32 give insight into RT growth rate for each response method, allowing performance projection as EPS increases. A consolidated model is also developed for performance estimates in dynamic environments.

6.3 Research Significance

SDN interest is exploding; its market value is forecast to grow at 86% Compound Annual Growth Rate (CAGR), from being valued at approximately $2 billion in 2015 to $132 billion+ in 2022 [58]. All this publicity about SDN begs the question - is it realistic? Google says yes. In fact, they have already implemented SDN into the backbone Wide Area Network (WAN) that connects their data centers all across the globe. By doing this, they have proven: 1) conversion from a traditional network is possible; and 2) the results are substantial. Their bandwidth utilization on their new SDN has been measured at 95%, a huge increase from traditional rates of 30-40% [59] [60]. As more technological leaders of the Open Networking Foundation continue to implement and promote SDN, it will become a standard practice in the industry.

As this trend continues, migration of network services must naturally follow with security as a top priority. Decisions will be made about maintaining a heterogeneous intrusion detection system (if any) or transitioning to what this research proposes: a unified SDN security management solution. There are advantages to the unique information both systems collect and can be extremely effective when they work together. Network protection *is* possible by authorizing a central controller to manage layer 2 devices via the OpenFlow protocol; however, it is imperative to provide the controller accurate security data, namely from a well-configured HIDS manager. Ideally, the privileged controller functionality is abstracted to a secure API, permitting the HIDS manager to initiate requests based off the present or historical data it already has. This abstracted version is exactly what the research represents through the AHNSR system.

6.4 Future Work

There are many possibilities of extending the AHNSR system due to additional research and/or tools developed in other domains. Incorporation of these tools could increase performance, scalability, usability, and more. The following suggests four future work options based off what was encountered while conducting this research:

- First, the limitation of actual "real-time" log collection can be solved by internal development in the OSSEC agent software. It is possible to modify the process by utilizing the *libevent* library, and configuring it to trigger upon new entries in the same log files monitored by OSSEC. Therefore, it could achieve real time monitoring based off triggers rather than timeout intervals. Since those modifications can affect overall performance, more evaluation is necessary to determine the significance.

- Second, there is more potential in utilizing the historical data stored in the OSSEC database. For the entirety OSSEC is running in a network, it is collecting data on each host's actions. Login times, syslog, and application interaction all put together effectively captures an individual's cyber behavior. One could implement machine learning with this data, taking advantage of the unprocessed training set naturally developed. Alerts could then be generated off a more intelligent algorithm rather than simple regex parameters. Trends can be developed on a per-hostname and/or per-username basis, giving more control and insight into possible insider threats as well.

- Third, there is always room for improvement in program execution, management, and abstraction. Regarding Floodlight, features are continually being developed that give users greater flexibility and control over their SDN. The OpenFlow protocol continues to be updated frequently but adopted in practice slowly, thereby making it difficult to utilize newer features, even if some of these may be more efficient/effective in handling appropriate active responses. Creativity and innovation are supreme though - there are endless possibilities for finite control over network and transport layer headers at each packet forwarding device in networks around the globe.

- Lastly, a different CPU utilization collection technique can be implemented. This research could not bypass the frequency limitation of per-second reports using the `mpstat` command. Another method with sub-second insight may be more effective in providing a higher sample size of CPU data, consequentially yielding more accurate results.

6.5 Chapter Summary

This chapter concludes the background, design, methodology, and results of this research. It summarizes the overall research conclusions, discusses the significance of the work, and suggests opportunities for future work in improving the AHNSR framework.

Appendix A. Log Generation Script

```
#include <windows.h>
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
using namespace std;

/// Create a Timer, which will immediately begin counting
/// up from 0.0 seconds.
/// You can call reset() to make it start over.
class Timer {
public:
  Timer() {
    ///GetSystemTimeAsFileTime(&timestart);
    GetSystemTime(&timestart);
    reset();
  }
  /// reset() makes the timer start over counting from 0.0 seconds.
  void reset() {
    unsigned __int64 pf;
    QueryPerformanceFrequency((LARGE_INTEGER *)&pf);
    freq_ = 1.0 / (double)pf;
    QueryPerformanceCounter((LARGE_INTEGER *)&baseTime_);
  }
  /// seconds() returns the number of seconds (to very high resolution)
  /// elapsed since the timer was last created or reset().
  double seconds() {
    unsigned __int64 val;
    QueryPerformanceCounter((LARGE_INTEGER *)&val);
    return (val - baseTime_) * freq_;
  }
  /// seconds() returns the number of milliseconds (to very high resolution)
  /// elapsed since the timer was last created or reset().
  double milliseconds() {
    return seconds() * 1000.0;
  }

  string getStartTime() {
    ///ULONGLONG t = ((ULONGLONG)timestart.dwHighDateTime << 32) | (ULONGLONG)
        timestart.dwLowDateTime;
    ///return (double)t / 10000000.0;
    string toReturn = to_string(timestart.wHour) + ":" + to_string(timestart.wMinute)
         + ":" + to_string(timestart.wSecond) + "." + to_string(timestart.
        wMilliseconds);
    return toReturn;
  }
private:
  double freq_;
  unsigned __int64 baseTime_;
  ///FILETIME timestart;
  SYSTEMTIME timestart;
};

int main(int argc, char* argv[]) {

  //Start Timer to syncronize with server (trigger)
  Timer myTimer;

  //Validate input
  if (argc != 3) {
    // Tell the user how to run the program
    cerr << "Usage: " << argv[0] << " <EVENTS-PER-SECOND> <SECONDS-TO-RUN>";
    return 1;
  }
  istringstream ss1(argv[1]);
  int EVENTS_PER_SECOND;
  if (!(ss1 >> EVENTS_PER_SECOND)) {
    cerr << "Invalid number: " << argv[1];
    return 1;
  }
  istringstream ss2(argv[2]);
  int SECONDS_TO_RUN;
  if (!(ss2 >> SECONDS_TO_RUN)) {
    cerr << "Invalid number: " << argv[2];
    return 1;
  }

  //Create File handle
  ofstream outfile;

  //Clear the current log
  //outfile.open("experimentlog.txt", ios_base::trunc);
  //outfile.close();

  //Generate X logs per second for Y seconds, based off input args
  //closing file after each write so it doesn't lock control
  for (int currentSecond = 0; currentSecond < SECONDS_TO_RUN; currentSecond++) {
    //If it isn't the next second yet, then sleep for 10 ms
    while (myTimer.seconds() < currentSecond) {
      Sleep(10);
    }
    for (int i = 1; i <= EVENTS_PER_SECOND; i++) {
      outfile.open("experimentlog.txt", ios_base::app);
      outfile << "EXPERIMENT1: an event occurred. ";
      outfile << "LEVEL=1 ";
      outfile << "ID=";
      outfile << (currentSecond * EVENTS_PER_SECOND + i); //accumulate event ID over
          the entire period
      outfile << " TIME=" << myTimer.seconds();
      outfile << "\n";
      outfile.close();
      // cout << myTimer.seconds();
      /// STRINGSTREAM METHOD??
      // stringstream tempString;
      // tempString << "echo EXPERIMENT1: event level 1 generated. IP:192.168.0.102
          ID:";
      // string tempCmd("echo EXPERIMENT1: event level 1 generated. IP:192.168.0.102
          ID:" + i + " >> experimentlog.txt");
      // system(tempCmd.c_str());
    }
  }

  //outfile << "Total Seconds passed: " << myTimer.seconds();
  //outfile << "\n";
  //outfile.close();

  return 0;
}
```

Appendix B. Experiment Management Script

```
#!/bin/bash
#Author Jon Goodgion

#Check for command line args
if [ $# -eq 0 ]
  then
    echo "NO ARGUMENTS SUPPLIED. RUN WITH <EXP NAME> <WAIT-TIME> <optional:TRIAL-
        NUMBER>"
    echo "EXP NAME formatted as HOST#-EPS#-customstring (e.g. 10-1-acltest)"
    exit 1
elif [ -z "$1" ]
  then
    echo "Need EXP NAME"
    exit 1
elif [ -z "$2" ]
  then
    echo "Need Wait time argument"
    exit 1
fi

if [ -z "$3" ]
  then
  trialnumber=1
else
  trialnumber="$3"
  echo "Executing with $trialnumber trials"
fi
expname="$1"

#Setup Java classpath
export CLASSPATH=/home/ubuntu/Downloads/mysql-connector-java-5.1.40/mysql-connector-
    java-5.1.40-bin.jar:$CLASSPATH

echo " --- Creating client execution script... ---"

IFS='-' read -r -a array <<< "$expname"
EPS="${array[1]}"
echo "./startLogger3.sh $EPS" >> ~/clientCommand.txt

sleeptime="$2"
echo "SLEEPTIME=$sleeptime seconds"
echo "EPS=$EPS"

#If a firewall experiment, start firewall module and default allow through the pica
    switch
if [[ $expname == *"firewall"* ]]
then
        echo "Enabling Firewall module and allowing flows through switch 5e:3e:c4
            :54:44:4f:2b:ba"
        curl http://localhost:8080/wm/firewall/module/enable/json -X PUT -d ''
        curl -X POST -d '{"switchid": "5e:3e:c4:54:44:4f:2b:ba"}' http://localhost
            :8080/wm/firewall/rules/json
else
  echo "Firewall module is DISABLED"
fi

# BEGIN FOR LOOP (TRIALS)
for i in `seq 1 $trialnumber`;
do

  echo "--- Resetting OSSEC server logs... ---"
  ncat 10.131.0.11 12344 --idle-timeout 1
  > /home/ubuntu/cpuUsage.log
  echo "--- Send experiment execution to clients ---"
  ncat 192.168.0.101 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.102 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.103 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.104 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.105 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.106 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.107 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.108 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.109 12346 --idle-timeout 1 < ~/clientCommand.txt &
  ncat 192.168.0.110 12346 --idle-timeout 1 < ~/clientCommand.txt &
  wait

  echo "--- Ping priming the hosts ---"
  #ping all clients to force switch to create necessary temporary forwarding flows (
      reducing delay)
  ping -c 2 192.168.0.3 2>&1 >/dev/null &
  ping -c 2 192.168.0.101 2>&1 >/dev/null &
  ping -c 2 192.168.0.102 2>&1 >/dev/null &
  ping -c 2 192.168.0.103 2>&1 >/dev/null &
  ping -c 2 192.168.0.104 2>&1 >/dev/null &
  ping -c 2 192.168.0.105 2>&1 >/dev/null &
  ping -c 2 192.168.0.106 2>&1 >/dev/null &
  ping -c 2 192.168.0.107 2>&1 >/dev/null &
  ping -c 2 192.168.0.108 2>&1 >/dev/null &
  ping -c 2 192.168.0.109 2>&1 >/dev/null &
  ping -c 2 192.168.0.110 2>&1 >/dev/null &
  wait

  #then trigger timestamps/actions to occur simultaneously
  echo "--- Trigger Experiment Start ---"
  ncat 192.168.0.101 12345 --idle-timeout 1 &
  ncat 192.168.0.102 12345 --idle-timeout 1 &
  ncat 192.168.0.103 12345 --idle-timeout 1 &
  ncat 192.168.0.104 12345 --idle-timeout 1 &
  ncat 192.168.0.105 12345 --idle-timeout 1 &
  ncat 192.168.0.106 12345 --idle-timeout 1 &
  ncat 192.168.0.107 12345 --idle-timeout 1 &
  ncat 192.168.0.108 12345 --idle-timeout 1 &
  ncat 192.168.0.109 12345 --idle-timeout 1 &
  ncat 192.168.0.110 12345 --idle-timeout 1 &
  ncat 192.168.0.3 12345 --idle-timeout 1 &

  #New version CPU logging
  mpstat 1 10 | grep -A 10 "%idle" | tail -n 10 | awk -F " " '{print $1 " " (100-$13)
      }' > /home/ubuntu/cpuUsage.log
  wait
  echo "--- Start Triggers complete ---"
  echo "--- Waiting for $sleeptime seconds... ---"
  sleep "$sleeptime"

  echo "--- Gathering Data... ---"
  filename="$expname$i"
  scp root@10.131.0.11:/start-times.txt ~/RESULTS/"$filename"-startTime.txt
  scp root@10.131.0.11:/cpuUsage.log ~/RESULTS/"$filename"-cpuUsage.txt
  scp root@10.131.0.11:/var/ossec/logs/active-responses.log ~/RESULTS/"$filename"-
      activeResponse.txt
  scp root@10.131.0.11:/var/ossec/logs/archives/archives.log ~/RESULTS/"$filename"-
      rawLogs.txt

  echo "--- Parsing Data... ---"

  cd ~/RESULTS
  if [[ $expname == *"echo"* ]]
  then
    echo using ECHO dataParser
    java dataParser "$expname"
  else
     echo using logger dataParser3
     java dataParser4 "$expname" "$i"
  fi

  #echo " --- Clearing all flows and ACL entries... ---"
  #curl http://localhost:8080/wm/acl/clear/json
  wait
  curl http://localhost:8080/wm/staticentrypusher/clear/all/json
  echo " "

done
#END FOR LOOP
```

```

#disable firewall module
if [[ $expname == *"firewall"* ]]
then
    echo "Disabling Firewall module..."
    curl http://localhost:8080/wm/firewall/module/disable/json -X PUT -d ''
fi


echo " --- Pushing to Drive... ---"
cp ~/RESULTS/"$expname"-cpu-data.csv ~/gdrive/
cp ~/RESULTS/"$expname"-data.csv ~/gdrive/
cd ~/gdrive
drive-google push --quiet --destination AFIT/Research/Results/ "$expname"-data.csv
drive-google push --quiet --destination AFIT/Research/Results/ "$expname"-cpu-data.
    csv
echo "$expname DONE"
```

Appendix C. OSSEC Configuration

```
<ossec_config>
    <global>
        <email_notification>no</email_notification>
        <logall>yes</logall>
    </global>
  <command>
      <name>static-entry</name>
      <executable>testcommand.sh</executable>
      <expect></expect>
    </command>
  <command>
        <name>logonly</name>
        <executable>logOnly.sh</executable>
        <expect></expect>
    </command>
  <command>
        <name>ACL-deny</name>
        <executable>ACL-deny.sh</executable>
        <expect></expect>
    </command>
  <command>
        <name>floodlight-firewall-deny</name>
        <executable>floodlight-firewall-deny.sh</executable>
        <expect></expect>
    </command>

  <active-response>
        <disabled>no</disabled>
        <command>ACL-deny</command>
        <location>server</location>
        <rules_group>experiment1</rules_group>
    </active-response>
  <active-response>
    <disabled>no</disabled>
      <command>logonly</command>
      <location>server</location>
      <rules_group>experiment1</rules_group>
  </active-response>
  <active-response>
        <disabled>no</disabled>
        <command>static-entry</command>
        <location>server</location>
        <rules_group>experiment1</rules_group>
    </active-response>
  <active-response>
        <disabled>no</disabled>
        <command>floodlight-firewall-deny</command>
        <location>server</location>
        <rules_group>experiment1</rules_group>
    </active-response>

    <rules>
        <include>rules_config.xml</include>
        <include>pam_rules.xml</include>
        <include>sshd_rules.xml</include>
        <include>telnetd_rules.xml</include>
        <include>syslog_rules.xml</include>
        <include>arpwatch_rules.xml</include>
        <include>symantec-av_rules.xml</include>
        <include>symantec-ws_rules.xml</include>
        <include>pix_rules.xml</include>
        <include>named_rules.xml</include>
        <include>smbd_rules.xml</include>
        <include>vsftpd_rules.xml</include>
        <include>pure-ftpd_rules.xml</include>
        <include>proftpd_rules.xml</include>
        <include>ms_ftpd_rules.xml</include>
        <include>ftpd_rules.xml</include>
        <include>hordeimp_rules.xml</include>
        <include>roundcube_rules.xml</include>
        <include>wordpress_rules.xml</include>
        <include>cimserver_rules.xml</include>
        <include>vpopmail_rules.xml</include>
        <include>vmpop3d_rules.xml</include>
        <include>courier_rules.xml</include>
        <include>web_rules.xml</include>
        <include>web_appsec_rules.xml</include>
        <include>apache_rules.xml</include>
        <include>nginx_rules.xml</include>
        <include>php_rules.xml</include>
        <include>mysql_rules.xml</include>
        <include>postgresql_rules.xml</include>
        <include>ids_rules.xml</include>
        <include>squid_rules.xml</include>
        <include>firewall_rules.xml</include>
        <include>apparmor_rules.xml</include>
        <include>cisco-ios_rules.xml</include>
        <include>netscreenfw_rules.xml</include>
        <include>sonicwall_rules.xml</include>
        <include>postfix_rules.xml</include>
        <include>sendmail_rules.xml</include>
        <include>imapd_rules.xml</include>
        <include>mailscanner_rules.xml</include>
        <include>dovecot_rules.xml</include>
        <include>ms-exchange_rules.xml</include>
        <include>racoon_rules.xml</include>
        <include>vpn_concentrator_rules.xml</include>
        <include>spamd_rules.xml</include>
        <include>msauth_rules.xml</include>
        <include>mcafee_av_rules.xml</include>
        <include>trend-osce_rules.xml</include>
        <include>ms-se_rules.xml</include>
        <include>zeus_rules.xml</include>
        <include>solaris_bsm_rules.xml</include>
        <include>vmware_rules.xml</include>
        <include>ms_dhcp_rules.xml</include>
        <include>asterisk_rules.xml</include>
        <include>ossec_rules.xml</include>
        <include>attack_rules.xml</include>
        <include>openbsd_rules.xml</include>
        <include>clam_av_rules.xml</include>
        <include>dropbear_rules.xml</include>
        <include>sysmon_rules.xml</include>
        <include>opensmtpd_rules.xml</include>
        <include>local_rules.xml</include>
    </rules>

    <rootcheck>
        <disabled>yes</disabled>
    </rootcheck>

    <global>
        <white_list>127.0.0.1</white_list>
        <white_list>::1</white_list>
        <white_list>^localhost.localdomain$</white_list>
        <white_list>10.1.2.2</white_list>
        <white_list>10.1.2.48</white_list>
    </global>

    <remote>
        <connection>secure</connection>
    </remote>

    <alerts>
        <log_alert_level>1</log_alert_level>
    </alerts>

    <command>
        <name>host-deny</name>
        <executable>host-deny.sh</executable>
        <expect>srcip</expect>
        <timeout_allowed>yes</timeout_allowed>
    </command>

    <command>
        <name>firewall-drop</name>
        <executable>firewall-drop.sh</executable>
        <expect>srcip</expect>
        <timeout_allowed>yes</timeout_allowed>
    </command>

    <command>
        <name>disable-account</name>
        <executable>disable-account.sh</executable>
        <expect>user</expect>
        <timeout_allowed>yes</timeout_allowed>
    </command>

    <command>
        <name>restart-ossec</name>
        <executable>restart-ossec.sh</executable>
        <expect></expect>
    </command>

    <command>
        <name>route-null</name>
        <executable>route-null.sh</executable>
        <expect>srcip</expect>
        <timeout_allowed>yes</timeout_allowed>
    </command>

<!-- Files to monitor (localfiles) -->

    <localfile>
        <log_format>syslog</log_format>
        <location>/var/log/auth.log</location>
    </localfile>

    <localfile>
        <log_format>syslog</log_format>
        <location>/var/log/syslog</location>
    </localfile>

    <localfile>
        <log_format>syslog</log_format>
        <location>/var/log/dpkg.log</location>
    </localfile>

    <localfile>
        <log_format>command</log_format>
        <command>df -P</command>
    </localfile>

    <localfile>
        <log_format>full_command</log_format>
        <command>netstat -tan |grep LISTEN |egrep -v '(127.0.0.1| ::1)' | sort</
            command>
    </localfile>

    <localfile>
        <log_format>full_command</log_format>
        <command>last -n 5</command>
    </localfile>

    <database_output>
        <hostname>localhost</hostname>
        <username>ossecuser</username>
        <password>ossec</password>
        <database>ossec</database>
        <type>mysql</type>
    </database_output>

</ossec_config>
```

Appendix D. Internal Options Configuration

```
# Analysisd default rule timeframe. Default 360
analysisd.default_timeframe=360
# Analysisd stats maximum diff.
analysisd.stats_maxdiff=999000
# Analysisd stats minimum diff. Default 150
analysisd.stats_mindiff=150
# Analysisd stats percentage (how much to differ from average) Default 150
analysisd.stats_percent_diff=150
# Analysisd FTS list size. Default 32
analysisd.fts_list_size=32
# Analysisd FTS minimum string size.
analysisd.fts_min_size_for_str=14
# Analysisd Enable the firewall log (at logs/firewall/firewall.log)
# 1 to enable, 0 to disable.
analysisd.log_fw=1

# Logcollector file loop timeout (check every 2 seconds for file changes)
logcollector.loop_timeout=2

# Logcollector number of attempts to open a log file.
logcollector.open_attempts=8

# Logcollector - If it should accept remote commands from the manager
logcollector.remote_commands=0

# Remoted counter io flush. Default 128
remoted.recv_counter_flush=128

# Remoted compression averages printout.
remoted.comp_average_printout=19999

# Verify msg id (set to 0 to disable it)
remoted.verify_msg_id=0

# Maild strict checking (0=disabled, 1=enabled)
maild.strict_checking=1

# Maild grouping (0=disabled, 1=enabled)
# Groups alerts within the same e-mail.
maild.groupping=1

# Maild full subject (0=disabled, 1=enabled)
maild.full_subject=0

# Maild display GeoIP data (0=disabled, 1=enabled)
maild.geoip=1

# Monitord day_wait. Ammount of seconds to wait before compressing/signing
# the files.
monitord.day_wait=10

# Monitord compress. (0=do not compress, 1=compress)
monitord.compress=1

# Monitord sign. (0=do not sign, 1=sign)
monitord.sign=1

# Monitord monitor_agents. (0=do not monitor, 1=monitor)
monitord.monitor_agents=1


# Syscheck checking/usage speed. To avoid large cpu/memory
# usage, you can specify how much to sleep after generating
# the checksum of X files. The default is to sleep 2 seconds
# after reading 15 files.
syscheck.sleep=2
syscheck.sleep_after=15


# Database - maximum number of reconnect attempts
```

```
dbd.reconnect_attempts=20

# Debug options.
# Debug 0 -> no debug
# Debug 1 -> first level of debug
# Debug 2 -> full debugging

# Windows debug (used by the windows agent)
windows.debug=0

# Syscheck (local, server and unix agent)
syscheck.debug=0

# Remoted (server debug)
remoted.debug=0

# Analysisd (server or local)
analysisd.debug=0

# Log collector (server, local or unix agent)
logcollector.debug=0

# Unix agentd
agent.debug=0

# EOF
```

Appendix E. Floodlight Properties

```
floodlight.modules=\
net.floodlightcontroller.jython.JythonDebugInterface,\
net.floodlightcontroller.storage.memory.MemoryStorageSource,\
net.floodlightcontroller.core.internal.FloodlightProvider,\
net.floodlightcontroller.threadpool.ThreadPool,\
net.floodlightcontroller.debugcounter.DebugCounterServiceImpl,\
net.floodlightcontroller.perfmon.PktInProcessingTime,\
net.floodlightcontroller.staticentry.StaticEntryPusher,\
net.floodlightcontroller.restserver.RestApiServer,\
net.floodlightcontroller.topology.TopologyManager,\
net.floodlightcontroller.routing.RoutingManager,\
net.floodlightcontroller.forwarding.Forwarding,\
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager,\
net.floodlightcontroller.ui.web.StaticWebRoutable,\
net.floodlightcontroller.loadbalancer.LoadBalancer,\
net.floodlightcontroller.firewall.Firewall,\
net.floodlightcontroller.simpleft.FT,\
net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl,\
net.floodlightcontroller.accesscontrollist.ACL,\
net.floodlightcontroller.statistics.StatisticsCollector
org.sdnplatform.sync.internal.SyncManager.authScheme=CHALLENGE_RESPONSE
org.sdnplatform.sync.internal.SyncManager.keyStorePath=/etc/floodlight/key2.jceks
org.sdnplatform.sync.internal.SyncManager.dbPath=/var/lib/floodlight/
org.sdnplatform.sync.internal.SyncManager.keyStorePassword=PassWord
org.sdnplatform.sync.internal.SyncManager.port=6009
org.sdnplatform.sync.internal.SyncManager.thisNodeId=1
org.sdnplatform.sync.internal.SyncManager.persistenceEnabled=FALSE
org.sdnplatform.sync.internal.SyncManager.nodes=[\
{"nodeId": 1, "domainId": 1, "hostname": "192.168.1.100", "port": 6642},\
{"nodeId": 2, "domainId": 1, "hostname": "192.168.1.100", "port": 6643}\
]
net.floodlightcontroller.forwarding.Forwarding.match=in-port, vlan, mac, ip,
    transport
net.floodlightcontroller.forwarding.Forwarding.detailed-match=src-mac, dst-mac, src-
    ip, dst-ip, src-transport, dst-transport
net.floodlightcontroller.forwarding.Forwarding.flood-arp=NO
net.floodlightcontroller.forwarding.Forwarding.idle-timeout=5
net.floodlightcontroller.forwarding.Forwarding.set-send-flow-rem-flag=FALSE
net.floodlightcontroller.forwarding.Forwarding.remove-flows-on-link-or-port-down=TRUE
net.floodlightcontroller.core.internal.FloodlightProvider.role=ACTIVE
net.floodlightcontroller.core.internal.FloodlightProvider.controllerId=1
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager.latency-history-
    size=10
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager.latency-update-
    threshold=0.5
net.floodlightcontroller.core.internal.FloodlightProvider.
    shutdownOnTransitionToStandby=true
net.floodlightcontroller.core.internal.OFSwitchManager.openFlowPort=6653
net.floodlightcontroller.core.internal.OFSwitchManager.openFlowAddresses=0.0.0.0
net.floodlightcontroller.core.internal.OFSwitchManager.workerThreads=16
net.floodlightcontroller.core.internal.OFSwitchManager.bossThreads=1
net.floodlightcontroller.core.internal.OFSwitchManager.connectionBacklog=1000
net.floodlightcontroller.core.internal.OFSwitchManager.connectionTimeoutMs=60000
net.floodlightcontroller.core.internal.OFSwitchManager.
    defaultMaxTablesToReceiveTableMissFlow=1
net.floodlightcontroller.core.internal.OFSwitchManager.
    maxTablesToReceiveTableMissFlowPerDpid={"00:00:00:00:00:00:00:01":"1","2":"1"}
net.floodlightcontroller.core.internal.OFSwitchManager.
    clearTablesOnInitialHandshakeAsMaster=YES
net.floodlightcontroller.core.internal.OFSwitchManager.
    clearTablesOnEachTransitionToMaster=YES
net.floodlightcontroller.core.internal.OFSwitchManager.keyStorePath=/home/ubuntu/
    floodlight-master/floodlight/switch-keystore.jks
net.floodlightcontroller.core.internal.OFSwitchManager.keyStorePassword=PassFL
net.floodlightcontroller.core.internal.OFSwitchManager.useSsl=YES
net.floodlightcontroller.core.internal.OFSwitchManager.supportedOpenFlowVersions=1.0,
     1.1, 1.2, 1.3, 1.4, 1.5
net.floodlightcontroller.core.internal.OFSwitchManager.switchesInitialState={"00
    :00:00:00:00:00:00:01":"ROLE_MASTER","00:00:00:00:00:00:00:02":"ROLE_MASTER", "00
    :00:00:00:00:00:00:03":"ROLE_MASTER", "00:00:00:00:00:00:00:04":"ROLE_MASTER","00
    :00:00:00:00:00:00:05":"ROLE_MASTER"}
```

```
net.floodlightcontroller.restserver.RestApiServer.keyStorePath=/home/ubuntu/
    floodlight-master/floodlight/rest-keystore.jks
net.floodlightcontroller.restserver.RestApiServer.keyStorePassword=changeit
net.floodlightcontroller.restserver.RestApiServer.httpsNeedClientAuthentication=YES
net.floodlightcontroller.restserver.RestApiServer.useHttps=YES
net.floodlightcontroller.restserver.RestApiServer.useHttp=NO
net.floodlightcontroller.restserver.RestApiServer.httpsPort=8081
net.floodlightcontroller.restserver.RestApiServer.httpPort=8080
net.floodlightcontroller.restserver.RestApiServer.accessControlAllowAllOrigins=FALSE
net.floodlightcontroller.statistics.StatisticsCollector.enable=FALSE
net.floodlightcontroller.statistics.StatisticsCollector.
    collectionIntervalPortStatsSeconds=10
net.floodlightcontroller.topology.TopologyManager.pathMetric=latency
net.floodlightcontroller.topology.TopologyManager.maxPathsToCompute=3
```

Appendix F. Active Response Script

```sh
#!/bin/sh
# Generates SDN Active Response
# Author: Jon Goodgion

# Save the bash call start time
BASHSTARTTIME=`date +%H:%M:%S.%N`

#Store local directory structure
LOCAL=`dirname $0`;
cd $LOCAL
cd ../
PWD=`pwd`

#Store command line args
ACTION=$1
USER=$2
ALERTID=$4
RULEID=$5

#Src-ip pulled from ossec location info
SRCIP=`echo "$7" | cut --delimiter='-' --fields=1`

#Global values and dummy destination IP
DSTIP="1.2.3.4"
ADDACTION="add"
DELACTION="delete"

# ACL ACTION
if [ "$ACTION" = "$ADDACTION" ]; then
  #Query Secure REST API
  ACLRESPONSE=`curl --cacert /rest-cacert.pem --cert /ossec-selfsigned.crt --key /ossec-selfsigned.key -X POST -d "{\"src-ip\":\"$SRCIP/32\",\"dst-ip\":\"$DSTIP/32\",\"action\":\"deny\"}" https://10.131.0.10:8081/wm/acl/rules/json`
  if [ "$ACLRESPONSE" = '{"status" : "Success! New rule added."}' ]; then
          echo "BASH-START:$BASHSTARTTIME ACL-ADD:`date +%H:%M:%S.%N` SRCIP:$SRCIP ALERTID:$4 RULEID:$5 AGENTNAME:$6" >> /var/ossec/logs/active-responses.log
  elif [ "$ACLRESPONSE" = '{"status" : "Failed! The new ACL rule matches an existing rule."}' ]; then
    echo "BASH-START:$BASHSTARTTIME ACL-MATCH:`date +%H:%M:%S.%N` SRCIP:$SRCIP ALERTID:$4 RULEID:$5 AGENTNAME:$6" >> /var/ossec/logs/active-responses.log
  else
    echo "ERROR: Couldn't add or test ACL entry" >> /var/ossec/logs/active-responses.log
  fi
elif [ "$ACTION" = "$DELACTION" ]; then
  ACLRULEID=`curl --cacert /rest-cacert.pem --cert /ossec-selfsigned.crt --key /ossec-selfsigned.key https://10.131.0.10:8081/wm/acl/rules/json | jq ".[] | select(.nw_src == \"$SRCIP/32\" and .nw_dst == \"$DSTIP/32\") | .id"`
  ACLRESPONSE=`curl --cacert /rest-cacert.pem --cert /ossec-selfsigned.crt --key /ossec-selfsigned.key -X DELETE -d "{\"ruleid\":\"$ACLRULEID\" }" https://10.131.0.10:8081/wm/acl/rules/json`
      if [ "$ACLRESPONSE" = '{"status" : "Success! Rule deleted"}' ]; then
              echo "ACL rule $ACLRULEID Deleted. SRCIP=$SRCIP DSTIP=$DSTIP ALERTID=$4 RULEID=$5 $6" >> /var/ossec/logs/active-responses.log
      else
    echo "ERROR: Couldn't delete ACL entry" >> /var/ossec/logs/active-responses.log
  fi
else
  echo "Comparisons failed" >> /var/ossec/logs/active-responses.log
fi

#FIREWALL ACTION
if [ "$ACTION" = "$ADDACTION" ]; then
  RESPONSE=`curl --cacert /rest-cacert.pem --cert /ossec-selfsigned.crt --key /ossec-selfsigned.key -X POST -d '{"src-ip": "192.168.0.109/32", "dst-ip": "192.168.0.103/32", "action":"deny"}' https://10.131.0.10:8081/wm/firewall/rules/json`
  if [[ $RESPONSE == *"added"* ]]; then
```

```bash
          echo "BASH-START:$BASHSTARTTIME FIREWALL-ADD:`date +%H:%M:%S.%N` SRCIP:$SRCIP ALERTID:$4 RULEID:$5 AGENTNAME:$6" >> /var/ossec/logs/active-responses.log
  elif [ "$RESPONSE" = '{"status" : "Error! A similar firewall rule already exists."}' ]; then
    echo "BASH-START:$BASHSTARTTIME FIREWALL-MATCH:`date +%H:%M:%S.%N` SRCIP:$SRCIP ALERTID:$4 RULEID:$5 AGENTNAME:$6" >> /var/ossec/logs/active-responses.log
  else
    echo "ERROR: Couldn't add firewall entry" >> /var/ossec/logs/active-responses.log
  fi
else
  echo "Add/Delete Action Comparisons failed. No curl attempted." >> /var/ossec/logs/active-responses.log
fi

#STATIC FLOW ACTION
if [ "$ACTION" = "$ADDACTION" ]; then
  #get the switch ID that the alerting host is connected to by querying Floodlight
  SWITCHID=`curl --cacert /rest-cacert.pem --cert /ossec-selfsigned.crt --key /ossec-selfsigned.key -s "https://10.131.0.10:8081/wm/device/?ipv4=$SRCIP" | jq -r '.devices[0].attachmentPoint[0].switch'`
  #SWITCHID="5e:3e:c4:54:44:4f:2b:ba"

  #Request flow insert for network response/reconfiguration vis secure REST Query:
  FLOWRESPONSE=`curl --cacert /rest-cacert.pem --cert /ossec-selfsigned.crt --key /ossec-selfsigned.key -X POST -d "{\"switch\":\"$SWITCHID\", \"name\":\"$ALERTID\", \"eth_type\":\"0x0800\", \"cookie\":\"0\", \"priority\":\"1\", \"in_port\":\"1\", \"active\":\"true\", \"actions\":\"output=1\"}" https://10.131.0.10:8081/wm/staticflowpusher/json`

  #Log successful flow entry in Active Response log
  if [ "$FLOWRESPONSE" = '{"status" : "Entry pushed"}' ]; then
    echo "BASH-START:$BASHSTARTTIME FLOW-ADD:`date +%H:%M:%S.%N` SRCIP:$SRCIP ALERTID:$4 RULEID:$5 AGENTNAME:$6" >> ${PWD}/../logs/active-responses.log
  fi

  #Delete the flow to allow subsequent ones to actually reach the switch
  curl -X DELETE -d '{"name":"EXPERIMENT"}' http://10.131.0.10:8080/wm/staticflowpusher/json
else
  echo "Add/Delete Action Comparisons failed. No logging attempted." >> /var/ossec/logs/active-responses.log
fi
```

Appendix G. Data Parsing

```java
import java.util.*;
import java.io.*;
import java.sql.*;

class dataParser4 {

  private static final String COMMA_DELIMITER = ",";
  private static final String NEW_LINE_SEPERATOR = "\n";
  private static final String COLUMN_HEADER = "trialnumber,eps,agent-id,alert-id,
      alert-level,start-time,start-time-offset,start-time-adjusted,end-time,
      difference,bashAR-start,difference-ARbash";

  public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    String fileNameArg = "";
    String outputFileName = "";
    String trialNumber = "";
    String numEPS = "";
    int logCount = 0;
    int rawlogTotalCount = 0;
    int rawlogExperimentCount = 0;
    if (args.length > 0) {
      try {
        trialNumber = args[1];
        fileNameArg = args[0] + trialNumber + "-";
        String[] filenametokens = fileNameArg.split("-");
        double numAgents = Double.parseDouble(filenametokens[0]);
        double epsPerAgent = Double.parseDouble(filenametokens[1]);
        numEPS = Double.toString(numAgents * epsPerAgent);
        outputFileName = args[0] + "-";
        //System.out.println("Filename argument: " + fileNameArg);
      } catch (Exception e) {
        System.err.println("Invalid argument. Expect test string (e.g. 2-100-1");
        System.exit(1);
      }
    }

    //Get starting time
    String startSeconds = "";
    try {
      String fullfile = fileNameArg + "startTime.txt";
      System.out.println("Reading file: " + fullfile);
      File file = new File(fileNameArg + "startTime.txt");
      Scanner input = new Scanner(file);
      String fullTime = input.next();
      String[] tokens = fullTime.split(":");
      startSeconds = tokens[2];
      System.out.println("Starting time trimmed: " + startSeconds);
    } catch (FileNotFoundException e) {
      System.err.println("File not found.");
      System.exit(1);
    }

    //Process everything else
    try {
      //Register mysql driver
      Class.forName("com.mysql.jdbc.Driver");

      //Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection("jdbc:mysql://192.168.0.3:3306/ossec","
          ossecuser", "ossec");

      //Build scanner for raw archive logs
      File rawLogs = new File(fileNameArg + "rawLogs.txt");
      Scanner rawLogScanner = new Scanner(rawLogs);

      //Count number of raw archived logs
      while(rawLogScanner.hasNext()) {
        rawlogTotalCount++;
        String nextRawLine = rawLogScanner.nextLine();
        String[] rawLogTokens = nextRawLine.split(" +|[=]");
        if(rawLogTokens.length == 21) {
          rawlogExperimentCount++;
        }
      }
      //Build scanner for AR logs
      File arLogs = new File(fileNameArg + "activeResponse.txt");
      Scanner arLogScanner = new Scanner(arLogs);

      System.out.println("Processings files: " + fileNameArg + "activeResponse.txt &
          " + fileNameArg + "rawLogs.txt");

      //Build headers on output csv file if it is the first trial
      FileWriter fileWriter = new FileWriter(outputFileName + "data.csv", true);
      if (trialNumber.equals("1")) {
        fileWriter.append(COLUMN_HEADER.toString());
        fileWriter.append(NEW_LINE_SEPERATOR);
      }

      //While there are still log entries
      while(arLogScanner.hasNext()) {
        logCount++;
        //Get and split AR log
        String nextARLogLine = arLogScanner.nextLine();
        String[] rawARTokens = nextARLogLine.split("[ :]");
        String flowAddTime = rawARTokens[7];
        String bashStartTime = rawARTokens[3];
        String alertID = rawARTokens[11];
        String agentId = rawARTokens[15].replace("(", "").replace(")", "");

        //Execute a sql query to get full log
        stmt = conn.createStatement();
        String sql;
        sql = "SELECT * FROM ossec.alert where alertid = '" + alertID + "'";
        ResultSet rs = stmt.executeQuery(sql);

        //Extract data from result set
        //Retrieve the entry
        rs.next();
        String full_log = rs.getString("full_log");

        //DEBUG: Display values
        //System.out.println("alertID=" + alertID + " -- " + full_log);

        //Clean-up environment
        rs.close();
        stmt.close();

        //Split the raw log
        String[] rawLogTokens = full_log.split(" +|[=]");

        //DEBUG token values
        //for (int i = 0; i < rawLogTokens.length; i++) {
        //  System.out.println("Token" + i + ": " + rawLogTokens[i]);
        //}

        String experimentid = rawLogTokens[12];
        String alertLevel = rawLogTokens[10];
        String timeoffset = rawLogTokens[14];

        //Calculate the adjusted start time
        double startTimeAdjustedDouble = Double.parseDouble(timeoffset);

        //Calculate the flow add time difference
        double timeDifference = Double.parseDouble(flowAddTime) -
            startTimeAdjustedDouble;
        //Compensate for wrapping on minute
        if(timeDifference < 0) {
          timeDifference = timeDifference + 60;
        }

        //Calculate the bash start time difference
        double timeDifferenceBash = Double.parseDouble(bashStartTime) -
            startTimeAdjustedDouble;
        //Compensate for wrapping on minute
        if(timeDifferenceBash < 0) {
          timeDifferenceBash = timeDifferenceBash + 60;
        }

        //Append all values to this row in csv
        fileWriter.append(trialNumber);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(numEPS);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(agentId);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(experimentid);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(alertLevel);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(startSeconds);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(timeoffset);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(Double.toString(startTimeAdjustedDouble));
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(flowAddTime);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(Double.toString(timeDifference));
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(bashStartTime);
        fileWriter.append(COMMA_DELIMITER);
        fileWriter.append(Double.toString(timeDifferenceBash));
        fileWriter.append(NEW_LINE_SEPERATOR);
      } //END - no more logs to process

      fileWriter.flush();
      fileWriter.close();
      conn.close();

      System.out.println("Success. Output to " + outputFileName + "data.csv");


      System.out.println("Processing cpu logs: " + fileNameArg + "cpuUsage.txt");
      //Build scanner for CPU files
      File osseccpuLog = new File(fileNameArg + "cpuUsage.txt");
      Scanner osseccpuLogScanner = new Scanner(osseccpuLog);

      File floodlightcpuLog = new File("/home/ubuntu/cpuUsage.log");
      Scanner floodlightcpuLogScanner = new Scanner(floodlightcpuLog);

      //Build csv writer
      FileWriter fileWriter2 = new FileWriter(outputFileName + "cpu-data.csv", true);
      if (trialNumber.equals("1")) {
        fileWriter2.append("time,ossec-cpu-usage,floodlight-cpu-usage");
        fileWriter2.append(NEW_LINE_SEPERATOR);
      }
      while(osseccpuLogScanner.hasNext() && floodlightcpuLogScanner.hasNext()) {
        //Get and split next cpu logs
        String nextOssecCpuLogLine = osseccpuLogScanner.nextLine();
        String[] rawOssecCpuTokens = nextOssecCpuLogLine.split("[ ]");
        String nextFloodlightCpuLogLine = floodlightcpuLogScanner.nextLine();
        String[] rawFloodlightCpuTokens = nextFloodlightCpuLogLine.split("[ ]");
        //Append all values to this row in csv
        fileWriter2.append(rawFloodlightCpuTokens[0]);
        fileWriter2.append(COMMA_DELIMITER);
        fileWriter2.append(rawOssecCpuTokens[1]);
        fileWriter2.append(COMMA_DELIMITER);
        fileWriter2.append(rawFloodlightCpuTokens[1]);
        fileWriter2.append(NEW_LINE_SEPERATOR);
      }
      fileWriter2.flush();
      fileWriter2.close();
      System.out.println("Success. Output to " + outputFileName + "cpu-data.csv");
      System.out.println(logCount + " active response logs");
      System.out.println(rawlogExperimentCount + " archived experiment raw logs");
      System.out.println(rawlogTotalCount + " archived total raw logs");

    } catch (FileNotFoundException e) {
      System.err.println("File not found.");
      System.exit(1);
    } catch (IOException e) {
      System.err.println("IO exception.");
      System.exit(1);
    } catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
    }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
    } finally{
      //finally block used to close resources
      try{
        if(stmt!=null)
          stmt.close();
        }catch(SQLException se2){
        }// nothing we can do
        try{
          if(conn!=null)
            conn.close();
        }catch(SQLException se){
        se.printStackTrace();
      }//end finally try
    }//end try

  }
}
```

Appendix H. Networking Parameters

```
# /etc/sysctl.conf
# Increase system file descriptor limit
fs.file-max = 10000

# Discourage Linux from swapping idle processes to disk (default = 60)
vm.swappiness = 10

# Increase ephermeral IP ports
net.ipv4.ip_local_port_range = 10000 65000

# Increase Linux autotuning TCP buffer limits
# Set max to 16MB for 1GE and 32M (33554432) or 54M (56623104) for 10GE, letting the
    kernel scale it based on RAM.
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.core.rmem_default = 16777216
net.core.wmem_default = 16777216
net.core.optmem_max = 40960
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216

# Make room for more TIME_WAIT sockets due to more clients,
# and allow them to be reused if we run out of sockets
# Also increase the max packet backlog
net.core.netdev_max_backlog = 50000
net.ipv4.tcp_max_syn_backlog = 30000
net.ipv4.tcp_max_tw_buckets = 2000000
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 10

# Disable TCP slow start on idle connections
net.ipv4.tcp_slow_start_after_idle = 0

# Also up the UDP limits
net.ipv4.udp_rmem_min = 8192
net.ipv4.udp_wmem_min = 8192

# Disable source routing and redirects
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.accept_source_route = 0

# Log packets with impossible addresses for security
net.ipv4.conf.all.log_martians = 1

# /etc/security/limits.conf
# allow all users to open 10000 files
* soft nofile 10000
* hard nofile 10000

# /etc/ssh/sshd_config
# ensure we consult pam
UsePAM yes

# /etc/pam.d/sshd
# ensure pam includes our limits
session required pam_limits.so
```

Appendix I. R Analysis Script: Tests 1-4

```r
library(ggplot2)
library(plyr)
library(colorspace)
library(RColorBrewer)
library(reshape2)

# -- PLOTTING FUNTIONS ---------------------------------------

#Creates scatter plot for the RT of each alert ID with multiple agents
#Points are colored by agent ID
createScatterPlotMulti <- function(dataset=NULL) {
  ggplot(dataset, aes(x=alert.id, y=difference)) +
      geom_point(shape=1, aes(color=agent.id)) +
      labs(size=4, x="Alert ID",y="Response Time (s)", color="Agent ID") +
      scale_color_brewer(palette = "Paired", labels = c("Agent 101", "Agent 102", "Agent 103", "Agent 104"
                                                            , "Agent 105", "Agent 106", "Agent 107", "Agent 108"
                                                            , "Agent 109", "Agent 110")) +
      theme_bw() +
      theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold"))
}

#Creates scatter plot of each alert ID for single agent
createScatterPlotSingle <- function(dataset=NULL) {
  ggplot(dataset, aes(x=alert.id, y=difference)) +
    geom_point(shape=1) +
    labs(size=4, x="Alert ID",y="Response Time (s)") +
    theme_bw() +
    theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold"))
}

#Creates scatter plot for the RT of each row in the dataset
#Index indicates the order processed in OSSEC
createScatterPlotIndex <- function(dataset=NULL) {
  ggplot(dataset, aes(x=index, y=difference)) +
    geom_point(shape=1) +
    ggtitle("Individual Alert Response Time") +
    labs(size=4, x="Alert Index",y="Response Time (s)") +
    theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold"),
          plot.title = element_text(size = rel(2)))
}
# -- ALERT ID / INDEX PLOTTING ---------------------------------------------------

#Read and create data frames
input = "1-250-logonly2"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\", filename, sep="")
results = read.csv(filepath)
createScatterPlotSingle(results)
input = "10-250-logonly"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\", filename, sep="")
results = read.csv(filepath)
createScatterPlotMulti(results)
input = "10-1000-logonly"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\", filename, sep="")
results = read.csv(filepath)

#Create index column
results$index = seq.int(nrow(results))
results$index = as.numeric(as.character(results$index))
```

```

#Plot by alert ID and index ID
createScatterPlotSingle(results)
createScatterPlotIndex(results)

# -- 500 EPS Datasets -----------------------------------------------------

#Read and create data frames - New data (3 Jan 17) for 500 EPS
input = "10-50-NEWfirewallTRIALS"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
firewallresults = read.csv(filepath)
input = "10-50-NEWstaticTRIALS"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
staticresults = read.csv(filepath)
input = "10-50-NEWlogTRIALS"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
logresults = read.csv(filepath)
input = "10-50-NEWaclTRIALS"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
aclresults = read.csv(filepath)
input = "10-10-NEWacl20TRIALS"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
aclresults = read.csv(filepath)
input = "10-50-NEWacl10a"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
aclresults = read.csv(filepath)

#Read and create data frames - Original data (13 Dec 16) for 500 EPS
input = "10-50-lognoss1TRIAL"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
logresults = read.csv(filepath)
input = "10-50-aclnoss1TRIAL"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
aclresults = read.csv(filepath)
input = "10-50-firewallnoss1TRIAL"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
firewallresults = read.csv(filepath)
input = "10-50-staticnoss1TRIAL"
filename = paste(input, "-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
staticresults = read.csv(filepath)

#Add rows to distingush type
logresults$type = "Log-Only"
aclresults$type = "ACL"
firewallresults$type = "Firewall"
staticresults$type = "Static Flow"

#Summary analysis on each type's RT measurement
logSE = summarySE(logresults, measurevar = "difference", groupvars = c("type"))
aclSE = summarySE(aclresults, measurevar = "difference", groupvars = c("type"))
firewallSE = summarySE(firewallresults, measurevar = "difference", groupvars = c("
    type"))
staticSE = summarySE(staticresults, measurevar = "difference", groupvars = c("type"))
```

```
#Combine the results
combinedresults = rbind(logresults, aclresults, firewallresults, staticresults)
combinedSE = rbind(logSE, aclSE, firewallSE, staticSE)

#Plot the mean response times with CI bars
ggplot(combinedSE, aes(x=type, y=difference)) +
  geom_errorbar(aes(ymin=difference-ci, ymax=difference+ci), width=.5) +
  geom_point() +
  labs(size=4, x="SDN Response Type",y="Response Time (s)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

#Pairwise T-tests on type
pairwise.t.test(combinedresults$difference, combinedresults$type, p.adjust.method = "
    holm")

# -- RESPONSE TIME PER TRIAL BOX PLOTS --------------------------------

#Boxplots per trial for each response type
ggplot(logresults, aes(factor(trialnumber), difference)) +
  geom_boxplot() +
  geom_jitter(alpha=.2, shape=16, size=1) +
  labs(size=4, x="Trial #",y="Response Time (s)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )
ggplot(aclresults, aes(factor(trialnumber), difference)) +
  geom_boxplot() +
  geom_jitter(alpha=.2, shape=16, size=1) +
  labs(size=4, x="Trial #",y="Response Time (s)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )
ggplot(firewallresults, aes(factor(trialnumber), difference)) +
  geom_boxplot() +
  geom_jitter(alpha=.2, shape=16, size=1) +
  labs(size=4, x="Trial #",y="Response Time (s)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )
ggplot(staticresults, aes(factor(trialnumber), difference)) +
  geom_boxplot() +
  geom_jitter(alpha=.2, shape=16, size=1) +
  labs(size=4, x="Trial #",y="Response Time (s)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

#Box Plot for each method, consolidated trials
ggplot(combinedresults, aes(x=type, y=difference)) +
  geom_boxplot() +
  geom_jitter(alpha=.1, shape=16, size=.5) +
  labs(size=4, x="SDN Response Type",y="Response Time (s)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

# - CPU DATA IMPORT AND SUMMARY ----------------------------------------

input = "10-50-lognossl1TRIAL"
filename = paste(input, "-cpu-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
cpulogresults = read.csv(filepath)

input = "10-50-aclnossl1TRIAL"
filename = paste(input, "-cpu-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
cpuaclresults = read.csv(filepath)
```

```
input = "10-50-firewallnossl TRIAL"
filename = paste(input, "-cpu-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
cpufirewallresults = read.csv(filepath)

input = "10-50-staticnossl TRIAL"
filename = paste(input, "-cpu-data.csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
cpustaticresults = read.csv(filepath)

cpulogresults$type = "Log-Only"
cpuaclresults$type = "ACL"
cpufirewallresults$type = "Firewall"
cpustaticresults$type = "Static Flow"

## Only include first second of each test
cpulogresults.firstten = cpulogresults[c(1,11,21,31,41,51,61,71,81,91),]
cpuaclresults.firstten = cpuaclresults[c(1,11,21,31,41,51,61,71,81,91),]
cpufirewallresults.firstten = cpufirewallresults[c(1,11,21,31,41,51,61,71,81,91),]
cpustaticresults.firstten = cpustaticresults[c(1,11,21,31,41,51,61,71,81,91),]

#Perform SE on each
cpulogSE = summarySE(cpulogresults.firstten, measurevar = "ossec.cpu.usage",
    groupvars = c("type"))
cpuaclSE = summarySE(cpuaclresults.firstten, measurevar = "ossec.cpu.usage",
    groupvars = c("type"))
cpufirewallSE = summarySE(cpufirewallresults.firstten, measurevar = "ossec.cpu.usage"
    , groupvars = c("type"))
cpustaticSE = summarySE(cpustaticresults.firstten, measurevar = "ossec.cpu.usage",
    groupvars = c("type"))

cpuresultscombined.ALL = rbind(cpulogresults, cpustaticresults, cpuaclresults,
    cpufirewallresults)
#cpuresultscombined.SUBSET = rbind(subset(cpulogresults, ossec.cpu.usage > .3),
    subset(cpuaclresults, ossec.cpu.usage > 1), subset(cpufirewallresults, ossec.cpu.
    usage > 1), subset(cpustaticresults, ossec.cpu.usage > 1))
cpuresultscombined.FIRSTTEN = rbind(cpulogresults.firstten, cpustaticresults.firstten
    , cpuaclresults.firstten, cpufirewallresults.firstten)
cpuOSSECcombinedSE = rbind(cpulogSE, cpuaclSE, cpufirewallSE, cpustaticSE)

## CI on means - first ten only OSSEC cpu
ggplot(cpuOSSECcombinedSE, aes(x=type, y=ossec.cpu.usage)) +
  geom_errorbar(aes(ymin=ossec.cpu.usage-ci, ymax=ossec.cpu.usage+ci), width=.5) +
  geom_point() +
  ggtitle("") +
  labs(size=4, x="SDN Response Type",y="CPU Utilization (%)") +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      , plot.title = element_text(size = rel(2)))

# - CPU DATA JITTER PLOTS (APPENDIX)
    -------------------------------------------------

## jitter plot ALL points OSSEC CPU
ggplot(cpuresultscombined.ALL, aes(x=type, y=ossec.cpu.usage)) +
  geom_point() +
  geom_jitter() +
  labs(size=4, x="SDN Response Type",y="CPU Utilization (%)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

## jitter plot ALL points Floodlight CPU
ggplot(cpuresultscombined.ALL, aes(x=type, y=floodlight.cpu.usage)) +
  geom_point() +
  geom_jitter() +
  labs(size=4, x="SDN Response Type",y="CPU Utilization (%)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

## jitter plot FIRST TEN points OSSEC CPU
```

```
ggplot(cpuresultscombined.FIRSTTEN, aes(x=type, y=ossec.cpu.usage)) +
  geom_point() +
  geom_jitter() +
  labs(size=4, x="SDN Response Type",y="CPU Utilization (%)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

## jitter plot FIRST TEN points Floodlight CPU
ggplot(cpuresultscombined.FIRSTTEN, aes(x=type, y=floodlight.cpu.usage)) +
  geom_point() +
  geom_jitter() +
  labs(size=4, x="SDN Response Type",y="CPU Utilization (%)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

# - CPU BOXPLOTS -------------------------------------------------

#Prep and reshape dataframe
cpuresultscombined.FIRSTTEN = rename(cpuresultscombined.FIRSTTEN, c("floodlight.cpu.
    usage"="Floodlight", "ossec.cpu.usage"="OSSEC"))
cpuresultscombined.FIRSTTEN = melt(cpuresultscombined.FIRSTTEN)
cpuresultscombined.FIRSTTEN = rename(cpuresultscombined.FIRSTTEN, c("variable"="
    server"))
cpuresultscombined.FIRSTTEN.SE = summarySE(cpuresultscombined.FIRSTTEN, measurevar =
    "value", groupvars = c("type", "server"))

### boxplot-bothservers-expected
ggplot(cpuresultscombined.FIRSTTEN, aes(x=type, y=value, color=server)) +
  geom_boxplot() +
  labs(size=4, x="SDN Response Type",y="CPU Utilization (%)",color="Server") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

## CI on means - both OSSEC and FLOODLIGHT
ggplot(cpuresultscombined.FIRSTTEN.SE, aes(x=type, y=value, color=server)) +
  geom_errorbar(aes(ymin=value-ci, ymax=value+ci), width=.5) +
  geom_point() +
  labs(size=4, x="SDN Response Type",y="CPU Utilization (%)",color="Server") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

## -- CPU ANOVA -----------------------------------------------

cpuresultsOSSECcombined = subset(cpuresultscombined.FIRSTTEN, cpuresultscombined.
    FIRSTTEN$server == "OSSEC")
cpuresultsFLOODLIGHTcombined = subset(cpuresultscombined.FIRSTTEN, cpuresultscombined
    .FIRSTTEN$server == "Floodlight")
anova(lm(value ~ type, data = cpuresultsOSSECcombined))
anova(lm(value ~ type, data = cpuresultsFLOODLIGHTcombined))
pairwise.t.test(cpuresultsOSSECcombined$value, cpuresultsOSSECcombined$type, p.adjust
    .method = "holm")
pairwise.t.test(cpuresultsFLOODLIGHTcombined$value, cpuresultsFLOODLIGHTcombined$type
    , p.adjust.method = "holm")

## -- POWER T-TEST  -----------------------------------------------

power.t.test(sd = .3667, delta = .02, power = .90)
```

Appendix J. R Analysis Script: Load Tests

```r
library(ggplot2)
library(plyr)

## Create Summary function
##  Summarizes data.
## Gives count, mean, standard deviation, standard error of the mean, and confidence
    interval (default 95%).
##   data: a data frame.
##   measurevar: the name of a column that contains the variable to be summariezed
##   groupvars: a vector containing names of columns that contain grouping variables
##   na.rm: a boolean that indicates whether to ignore NA's
##   conf.interval: the percent range of the confidence interval (default is 95%)
##  AUTHOR: http://www.cookbook-r.com/Manipulating_data/Summarizing_data/
summarySE <- function(data=NULL, measurevar, groupvars=NULL, na.rm=FALSE, conf.
    interval=.95, .drop=TRUE) {

  # New version of length which can handle NA's: if na.rm==T, don't count them
  length2 <- function (x, na.rm=FALSE) {
    if (na.rm) sum(!is.na(x))
    else       length(x)
  }

  # This does the summary. For each group's data frame, return a vector with
  # N, mean, and sd
  datac <- ddply(data, groupvars, .drop=.drop,
                 .fun = function(xx, col) {
                   c(N    = length2(xx[[col]], na.rm=na.rm),
                     mean = mean   (xx[[col]], na.rm=na.rm),
                     sd   = sd     (xx[[col]], na.rm=na.rm)
                   )
                 },
                 measurevar
  )

  # Rename the "mean" column
  datac <- rename(datac, c("mean" = measurevar))

  datac$se <- datac$sd / sqrt(datac$N)  # Calculate standard error of the mean

  # Confidence interval multiplier for standard error
  # Calculate t-statistic for confidence interval:
  # e.g., if conf.interval is .95, use .975 (above/below), and use df=N-1
  ciMult <- qt(conf.interval/2 + .5, datac$N-1)
  datac$ci <- datac$se * ciMult

  return(datac)
}

#Given data frame, returns plot of response time difference vs EPS
createPlotEpsxDiff <- function(dataset=NULL, labeltext=NULL, xpos=0, ypos=0) {
  ggplot(dataset, aes(x=eps, y=difference)) +
    geom_errorbar(aes(ymin=difference-ci, ymax=difference+ci), width=5) +
    geom_line(col="blue", size=1) +
    geom_smooth(method=lm , color="red", se=TRUE) +
    geom_text(data = NULL, x = xpos, y = ypos, label=labeltext) +
    geom_point() +
    geom_ribbon(aes(ymin=difference-ci, ymax=difference+ci), alpha=0.2) +
    labs(size=4, x="Events Per Second",y="Response Time (s)") +
    theme_bw() +
    theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold
        "))
}

#Given consolidated data frame with every response type, plots time difference vs EPS
createPlotALL <- function(dataset=NULL) {
  ggplot(dataset, aes(x=eps, y=difference, fill=type, linetype=type)) +
    geom_errorbar(aes(ymin=difference-ci, ymax=difference+ci), width=5) +
    geom_line() +
    geom_point() +
    geom_ribbon(aes(ymin=difference-ci, ymax=difference+ci), alpha=0.2) +
    labs(size=4, x="Events Per Second",y="Response Time (s)",fill="Type",linetype="
        Type") +
    theme_bw() +
    theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold
        "))
}

# - NEW LOG ONLY -------------------------------------------------------

log.10.1 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-1-NEWlog20TRIALS-data.csv")
#log.10.10 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-10-logonly-data.csv")
#log.10.100 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-100-logonly-data.csv")
log.10.10 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-10-NEWlog20TRIALS-data.csv")
log.10.100 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-100-NEWlog10TRIALS-data.csv")
log.10.250 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-250-logonly-data.csv")
log.10.500 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-500-logonly-data.csv")
log.10.750 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-750-logonly-data.csv")
log.10.1000 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-1000-logonly-data.csv")

logCombined = rbind(log.10.1000, log.10.750, log.10.500, log.10.250, log.10.100, log
    .10.10, log.10.1)
logCombined$type = "Log-Only"

logStressSE = summarySE(logCombined, measurevar = "difference", groupvars = c("type",
    "eps"))

createPlotEpsxDiff(dataset = logStressSE)

# - FIREWALL ---------------------------------------------------------

firewall.10.1 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-1-NEWfirewall20TRIALSSS-data.csv")
#firewall.10.10 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-10-firewallnossl2-data.csv")
firewall.10.100 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-100-firewallssl3-data.csv")
firewall.10.10 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-10-NEWfirewall20TRIALSS-data.csv")
#firewall.10.100 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-100-NEWfirewall10TRIALSS-data.csv")
#firewall.10.250 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-250-firewallssl-data.csv")
firewall.10.250 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-250-NEWfirewall5TRIALS-data.csv")
firewall.10.500 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-500-firewallssl-data.csv")
firewall.10.750 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-750-firewall3-data.csv")
firewall.10.1000 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-1000-firewall-data.csv")

# fix EPS column
firewall.10.10$eps = 100
firewall.10.100$eps = 1000
firewall.10.250$eps = 2500
firewall.10.500$eps = 5000
firewall.10.750$eps = 7500
firewall.10.1000$eps = 10000
#firewall.10.2500$eps = 25000

firewallCombined = rbind(firewall.10.1000, firewall.10.750, firewall.10.500, firewall
    .10.250, firewall.10.100, firewall.10.10, firewall.10.1)
firewallCombined$type = "Firewall"

```

```
firewallStressSE = summarySE(firewallCombined, measurevar = "difference", groupvars =
     c("type","eps"))
#firewallStressBashSE = summarySE(firewallCombinedSE, measurevar = "difference.ARbash
    ", groupvars = "eps")

createPlotEpsxDiff(dataset=firewallStressSE)

# - ACL -------------------------------------------------------

acl.10.1 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-1-NEWacl20TRIALS-data.csv")
#acl.10.10 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-10-aclnossl-data.csv")
#acl.10.100 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-100-aclnossl-data.csv")
acl.10.10 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-10-NEWacl20TRIALS-data.csv")
acl.10.100 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-100-NEWacl10TRIALSS-data.csv")
acl.10.250 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-250-aclnossl-data.csv")
acl.10.500 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-500-aclnossl-data.csv")
acl.10.750 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-750-aclnossl-data.csv")
acl.10.1000 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-1000-aclnossl-data.csv")
#acl.10.2500 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-2500-firewall-data.csv")

aclCombined = rbind(acl.10.1000, acl.10.750, acl.10.500, acl.10.250, acl.10.100, acl
    .10.10, acl.10.1)
aclCombined$type = "ACL"

aclStressSE = summarySE(aclCombined, measurevar = "difference", groupvars = c("type",
    "eps"))
#aclStressBashSE = summarySE(aclCombined, measurevar = "difference.ARbash", groupvars
     = "eps")

createPlotEpsxDiff(dataset = aclStressSE)

# - STATIC FLOW -------------------------------------------------------

static.10.1 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-1-NEWstatic20TRIALS-data.csv")
#static.10.10 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-10-staticflownossl-data.csv")
#static.10.100 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-100-staticflownossl-data.csv")
static.10.10 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-10-NEWstatic20TRIALS-data.csv")
static.10.100 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-100-NEWstatic10TRIALS-data.csv")
#static.10.250 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-250-staticflownossl-data.csv")
#static.10.250 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-250-NEWstatic5TRIALS-data.csv")
static.10.250 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-250-NEWstatic1TRIAL-data.csv")
static.10.500 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-500-staticflownossl-data.csv")
static.10.750 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results
    \\10-750-staticflownossl-data.csv")
static.10.1000 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-1000-staticflownossl-data.csv")

staticCombined = rbind(static.10.1000, static.10.750, static.10.500, static.10.250,
    static.10.100, static.10.10, static.10.1)
staticCombined$type = "Static Flow"

staticStressSE = summarySE(staticCombined, measurevar = "difference", groupvars = c("
    type","eps"))
createPlotEpsxDiff(dataset = staticStressSE)
```

```

# - ALL TYPES COMBINED --------------------------------------------------------

#test boxplot
loadtest.ALL.RAW = rbind(staticCombined, logCombined, firewallCombined, aclCombined)
loadtest.ALL.RAW$eps = as.factor(loadtest.ALL.RAW$eps)
ggplot(loadtest.ALL.RAW, aes(x=eps, y=difference, fill=type)) +
  geom_boxplot() +
  ggtitle("") +
  labs(size=4, x="EPS Level",y="Response Time (s)") +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      , plot.title = element_text(size = rel(2)))

#Line plots RT all combined load test
allCombinedSE = rbind(aclStressSE, firewallStressSE, staticStressSE, logStressSE)
createPlotALL(dataset = allCombinedSE)

# Calculating the biggest gap
allCombinedSE[12, "difference"] - allCombinedSE[12, "ci"] - allCombinedSE[24, "
    difference"] + allCombinedSE[24, "ci"]


# --- CPU ALL ----------------------------------------
input = "cpu-all"
filename = paste(input, ".csv", sep="")
filepath = paste("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\Results\\",
    filename, sep="")
cpuALL = read.csv(filepath)

ggplot(cpuALL, aes(x=eps, y=cpu.ossec, fill=type, linetype=type, color=type)) +
  ylim(0,100) +
  geom_line() +
  geom_point() +
  labs(size=4, x="Events Per Second",y="CPU Utilization (%)",fill="Type", linetype="
      Type", color="Type") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      )

# - REGRESSION MODELS  --------------------------------------------------------

regressionACL = lm(difference ~ eps, data = aclStressSE)
regressionFIREWALL = lm(difference ~ eps, data = firewallStressSE)
regressionLOG = lm(difference ~ eps, data = logStressSE)
regressionSTATIC = lm(difference ~ eps, data = staticStressSE)

coeffACL = round(regressionACL$coefficients , 5)
modelText = paste("Model : ", coeffACL[1] , " + " , coeffACL[2] , "*x" , "\n\n" , "P-
    value adjusted = ",round(summary(regressionACL)$adj.r.squared,2))
createPlotEpsxDiff(dataset = aclStressSE, labeltext = modelText, xpos = 2500, ypos =
    3)

coeffFIREWALL = round(regressionFIREWALL$coefficients , 5)
modelText = paste("Model : ", coeffFIREWALL[1] , " + " , coeffFIREWALL[2] , "*x" , "\
    n\n" , "P-value adjusted = ",round(summary(regressionFIREWALL)$adj.r.squared,2))
createPlotEpsxDiff(dataset = firewallStressSE, labeltext = modelText, xpos = 2500,
    ypos = 6)

coeffLOG = round(regressionLOG$coefficients , 5)
modelText = paste("Model : ", coeffLOG[1] , " + " , coeffLOG[2] , "*x" , "\n\n" , "P-
    value adjusted = ",round(summary(regressionLOG)$adj.r.squared,2))
createPlotEpsxDiff(dataset = logStressSE, labeltext = modelText, xpos = 2500, ypos =
    1.1)

coeffSTATIC = round(regressionSTATIC$coefficients , 5)
modelText = paste("Model : ", coeffSTATIC[1] , " + " , coeffSTATIC[2] , "*x" , "\n\n"
     , "P-value adjusted = ",round(summary(regressionSTATIC)$adj.r.squared,2))
createPlotEpsxDiff(dataset = staticStressSE, labeltext = modelText, xpos = 2500, ypos
     = 3.5)

confint(regressionACL)
confint(regressionSTATIC)
confint(regressionLOG)
confint(regressionFIREWALL)

#combined model
loadtest.ALL.RAW$type = as.factor(loadtest.ALL.RAW$type)
allRAWSE = summarySE(loadtest.ALL.RAW, measurevar = "difference", groupvars = c("eps"
    ))
regressionALLRAW = lm(difference ~ eps * type - 1, data = loadtest.ALL.RAW)
summary(regressionALLRAW)
coeffALLRAW = round(regressionALLRAW$coefficients , 5)
modelText = paste("Model : ", coeffALLRAW[1] , " + " , coeffALLRAW[2] , "*x" , "\n\n"
     , "P-value adjusted = ",round(summary(regressionALLRAW)$adj.r.squared,2))
ggplot(loadtest.ALL.RAW, aes(x=eps, y=difference, group = 1)) +
  geom_smooth(method=lm, formula = loadtest.ALL.RAW$difference ~ loadtest.ALL.RAW$eps
       + loadtest.ALL.RAW$type - 1, color="red", se=TRUE) +
  ggtitle("") +
  labs(size=4, x="EPS Level",y="Response Time (s)") +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      , plot.title = element_text(size = rel(2)))

#plotting the sample custom function
fun.1 <-    function(x) .8 * (.42869 + .00007 * x) + .05 * (.46778 + .00033 * x) +
    .05 * (.23732 + .00059 * x) + .1 * (.49669 + .00027 * x)
fun.high <- function(x) .8 * (.52737 + .00009 * x) + .05 * (.85834 + .00040 * x) +
    .05 * (.98458 + .00072 * x) + .1 * (.68054 + .00030 * x)
fun.low <-  function(x) .8 * (.33000 + .00005 * x) + .05 * (.07721 + .00024 * x) +
    .05 * (-.50995 + .00045 * x) + .1 * (.31284 + .00025 * x)

ggplot(data.frame(x = 0), aes(x = x)) +
  stat_function(fun = fun.1) +
  xlim(0, 10000) +
  ylim(0.0, 2.5) +
  labs(size=4, x="EPS Level",y="Response Time (s)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      , legend.position="none")


#Inputting and plotting the dynamic test data
dynamic.10.500 = read.csv("C:\\Users\\jgoodgion\\Google Drive\\AFIT\\Research\\
    Results\\10-500-NEWdynamic2-data.csv")
dynamicSE = summarySE(dynamic.10.250, measurevar = "difference", groupvars = c("eps")
    )

ggplot(data.frame(x = 0), aes(x = x)) +
  stat_function(fun = fun.1) +
  stat_function(fun = fun.high, aes(colour = "red")) +
  stat_function(fun = fun.low, aes(colour = "red")) +
  geom_point(aes(x = 5000, y = 1.04), shape=4, color="blue", size=5) +
  xlim(0, 10000) +
  ylim(0.0, 2.5) +
  labs(size=4, x="EPS Level",y="Response Time (s)") +
  theme_bw() +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      , legend.position="none")


# - INDIVIDUAL ALERT MODELS -----------------------------------------------------------

static.10.750$index = seq.int(nrow(static.10.750))
ggplot(static.10.750, aes(x=index, y=difference)) +
  geom_point(size=1) +
  ggtitle("Linear Model of Increasing Individual Alerts") +
  xlim(0,10000) +
  ylim(0, 6) +
  labs(size=4, x="Alert ID",y="Response Time (s)") +
  geom_smooth(method=lm, formula=y~x, color="red", fullrange=TRUE) +
  theme(axis.text=element_text(size=12), axis.title=element_text(size=14,face="bold")
      , plot.title = element_text(size = rel(2)))
```

Appendix K. Trial Ranges

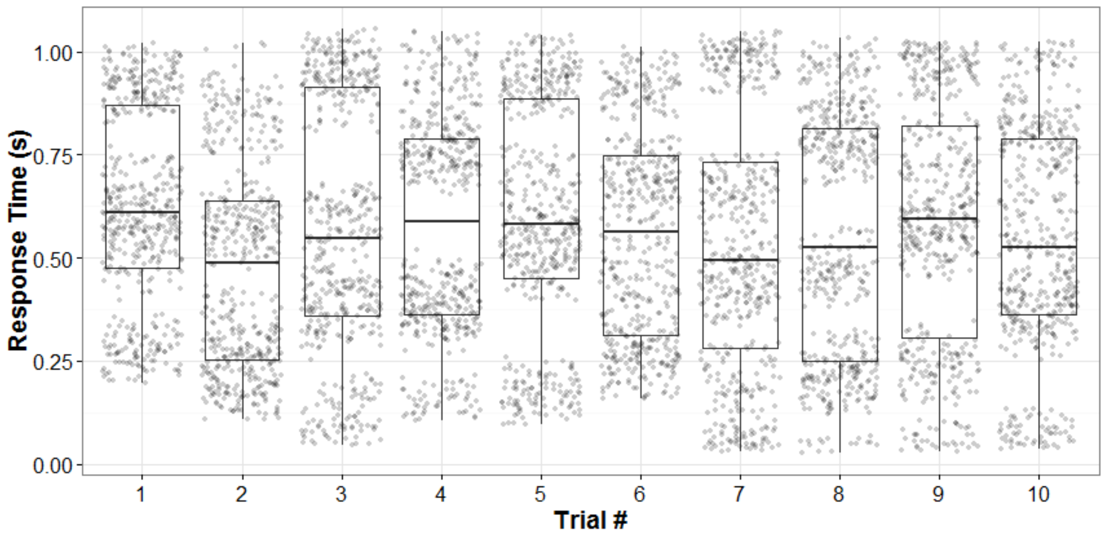


Figure 34. Static Flow RT quartile ranges (per trial at 500 EPS)

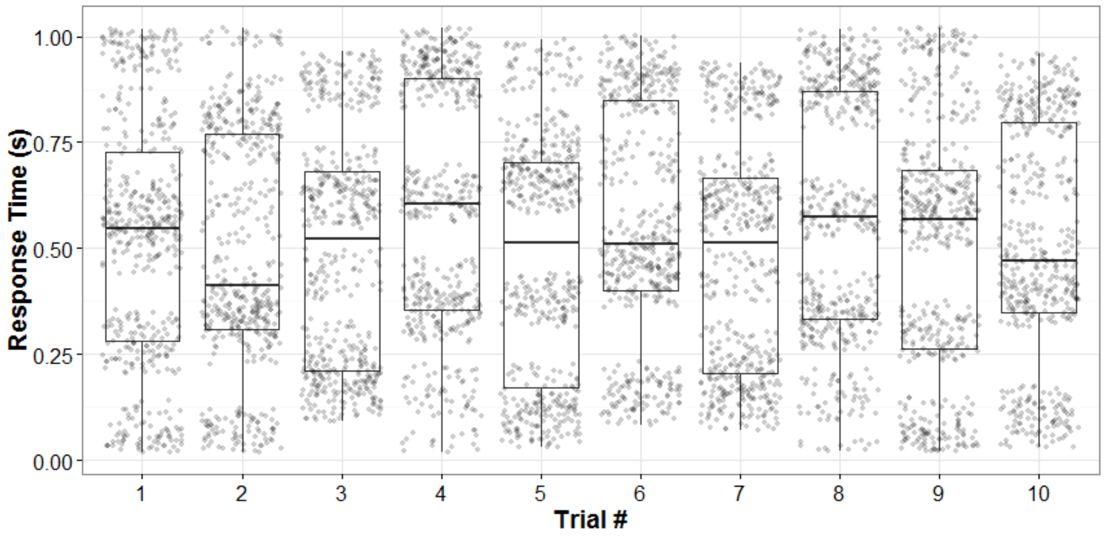


Figure 35. Log-Only RT quartile ranges (per trial at 500 EPS)

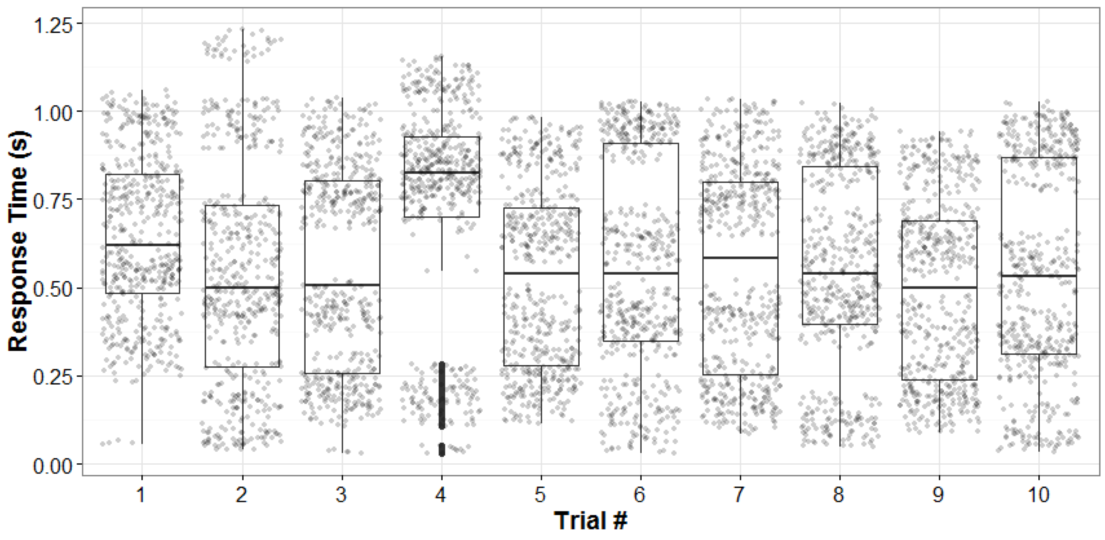


Figure 36. Firewall RT quartile ranges (per trial at 500 EPS)

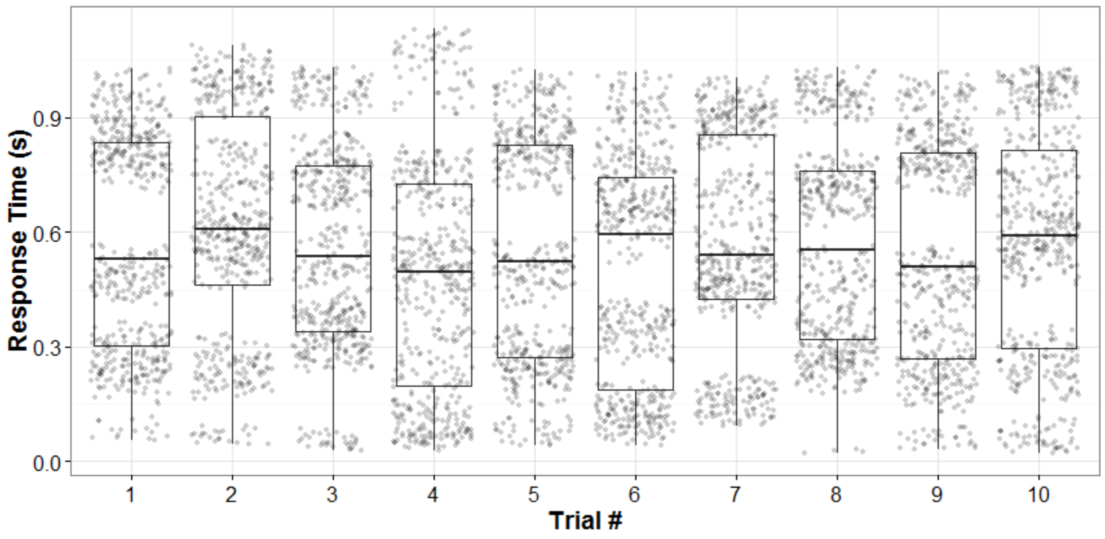


Figure 37. ACL RT quartile ranges (per trial at 500 EPS)

Appendix L. CPU Data Reduction

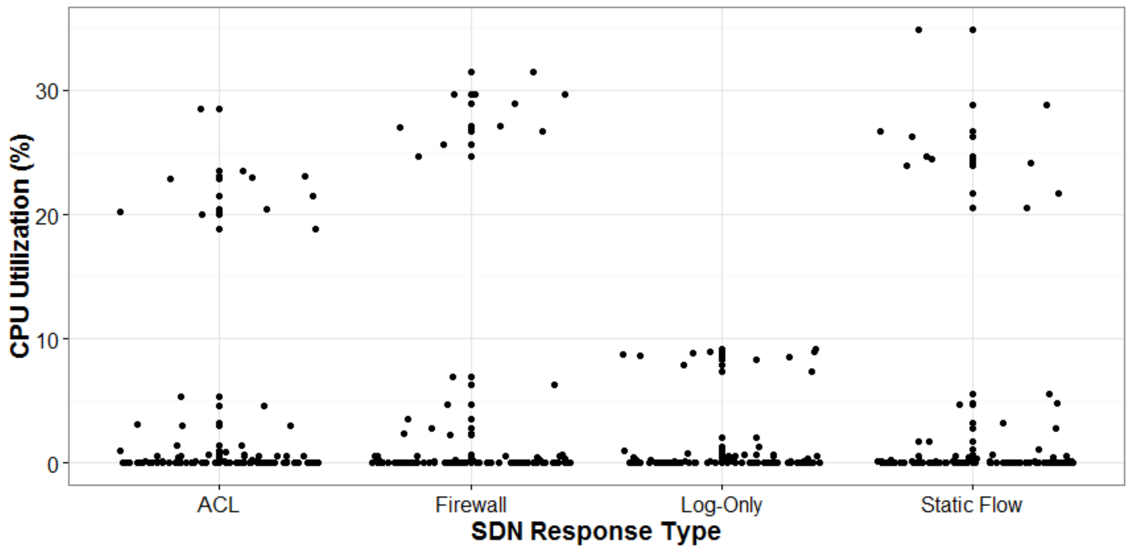


Figure 38. All OSSEC CPU utilization measurements (500 EPS)

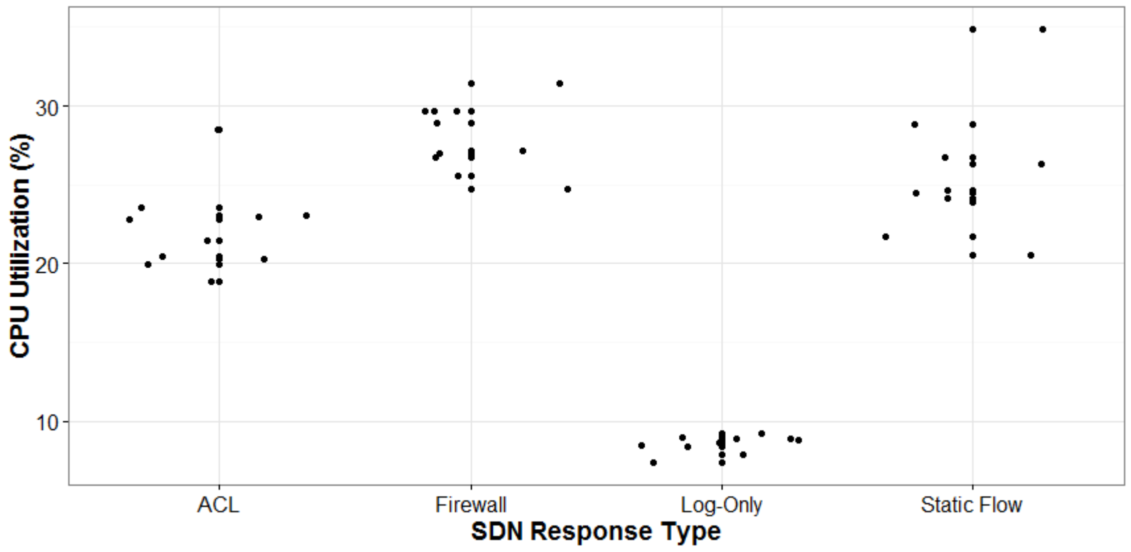


Figure 39. Reduced OSSEC CPU utilization measurements (500 EPS)

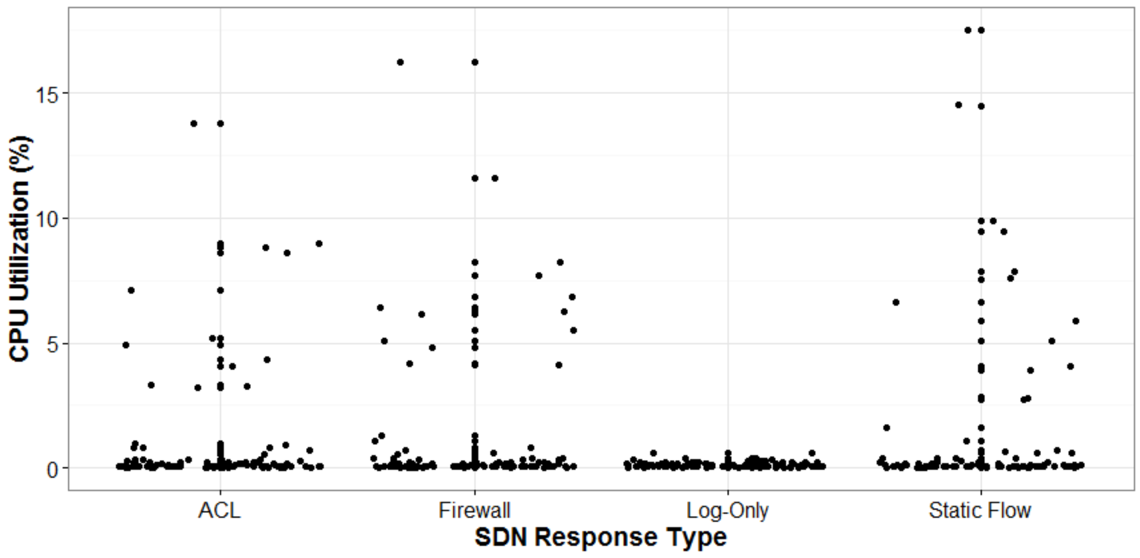


Figure 40. All Floodlight CPU utilization measurements (500 EPS)

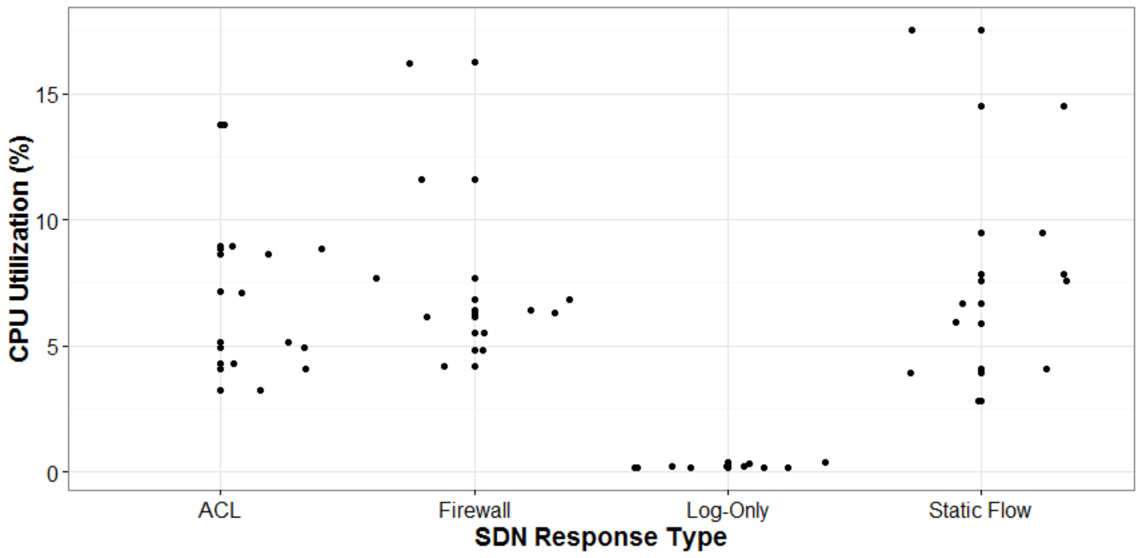


Figure 41. Reduced Floodlight CPU utilization measurements (500 EPS)

Appendix M. Threat Level Classification

Table 11. Internal OSSEC threat level classification descriptors [7]

Level	Description
01	None
02	System low priority notification - System notification or status messages. They have no security relevance.
03	Successful/Authorized events - They include successful login attempts, firewall allow events, etc.
04	System low priority error - Errors related to bad configurations or unused devices/applications. They have no security relevance and are usually caused by default installations or software testing.
05	User generated error - They include missed passwords, denied actions, etc. By itself they have no security relevance.
06	Low relevance attack - They indicate a worm or a virus that have no affect to the system (like code red for apache servers, etc). They also include frequently IDS events and frequently errors.
07	Bad word matching. They include words like bad, error, etc. These events are most of the time unclassified and may have some security relevance.
08	First time seen - Include first time seen events. First time an IDS event is fired or the first time an user logged in. If you just started using OSSEC HIDS these messages will probably be frequently. After a while they should go away, It also includes security relevant actions (like the starting of a sniffer or something like that).
09	Error from invalid source - Include attempts to login as an unknown user or from an invalid source. May have security relevance (specially if repeated). They also include errors regarding the admin (root) account.
10	Multiple user generated errors - They include multiple bad passwords, multiple failed logins, etc. They may indicate an attack or may just be that a user just forgot his credencials.
11	Integrity checking warning - They include messages regarding the modification of binaries or the presence of rootkits (by rootcheck). If you just modified your system configuration you should be fine regarding the syscheck messages. They may indicate a successful attack. Also included IDS events that will be ignored (high number of repetitions).
12	High importancy event - They include error or warning messages from the system, kernel, etc. They may indicate an attack against a specific application.
13	Unusual error (high importance) - Most of the times it matches a common attack pattern.
14	High importance security event. Most of the times done with correlation and it indicates an attack.
15	Severe attack - No chances of false positives. Immediate attention is necessary.

Bibliography

1. D. Kreutz, F. M.V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

2. N. McKeown. Making SDNs Work. Retrieved on October 1, 2016. URL *http://yuba.stanford.edu/~nickm/talks/ONS_2012.ppt*.

3. S. Hernan, S. Lambert, and T. Ostwald. Uncover Security Design Flaws using The STRIDE Approach. *MSDN Magazine-Louisville*, (1):1–8, 2006.

4. B. Lhotsky. *Instant OSSEC Host-based Intrusion Detection*. Packt Publishing Ltd, Birmingham, UK, 2013.

5. Pivotal. RabbitMQ Features. Retrieved June 1, 2016. URL *https://www.rabbitmq.com/features.html*.

6. M. Todd. Dynamic Network Security Control. Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, USA, 2015. URL *https://www.afit.edu/docs/AFIT-ENG-MS-16-M-049.pdf*.

7. R. Izard. Project Floodlight Documentation. Retrieved August 25, 2016. URL *https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+Documentation*.

8. Symantec. Internet Security Threat Report 2016. Technical report, 2016. URL *https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf*.

9. Verizon. 2016 Data Breach Investigations Report. Technical report, 2016. URL *http://www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf*.

10. PCI Security Standards Council. PCI DSS v3.1 PCI Quick Reference Guide. Retrieved December 20, 2016. URL *https://www.pcisecuritystandards.org/documents/PCIDSS_QRGv3_1.pdf*.

11. Y. Rechtman and K. Rashbaum. HIPAA security rule - Demystified. *The CPA Journal*, pages 68–70, 2015.

12. H. Farhady, H. Lee, and A. Nakao. Software-Defined Networking: A survey. *Computer Networks*, 81:79–95, 2015.

13. M. Mehta and B. Ives. Storage Area Networks. *Communications of the Association for Information Systems*, 14(1):70–71, 2004.

14. M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. *Sigcomm '07*, pages 1–12, 2007.

15. Open Networking Foundation. Software-Defined Networking: The New Norm for Networks [white paper], 2012. URL *https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf*.

16. P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, and B. Lantz. ONOS: towards an open, distributed SDN OS. *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, pages 1–6, 2014.

17. F. Long, Z. Sun, Z. Zhang, H. Chen, and L. Liao. Research on TCAM-based Openflow switch platform. In *2012 International Conference on Systems and Informatics, ICSAI 2012*, pages 1218–1221, 2012.

18. H. Yin, H. Xie, T. Tsou, P. Aranda, D. Lopez, and R. Sidi. SDNi: A Message Exchange Protocol for Software Defined Networks (SDNs). *Internet Research Task Force*, pages 1–14, 2012.

19. B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, A. Networks, and M. Casado. The Design and Implementation of Open vSwitch. *12th USENIX Symposium of Networked Systems Design and Implementation*, pages 117–130, 2015.

20. T. Elhourani, S. Ramasubramanian, and A. Kvalbein. IP Fast Rerouting for Multi-Link Failure. In *Networking IEEE/ACM Transactions*, pages 3014–3025, 2016.

21. T-Mobile. Video Streaming Without Using Your 4G LTE Data. Retrieved January 4, 2017. URL *https://www.t-mobile.com/offer/binge-on-streaming-video.html*.

22. E. Etherton. Eli the Computer Guy Collection. Retrieved December 1, 2015. URL *https://www.udemy.com/eli-the-computer-guy-collection/index.html*.

23. S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2):136–141, 2013.

24. M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. Van Reijendam, P. Weissmann, and N. McKeown. Maturing of OpenFlow and Software-defined Networking through deployments. *Computer Networks*, 61:151–175, 2014.

25. Z. Michael, D. Allan, M. Cohn, N. Damouny, C. Kolias, J. Maguire, S. Manning, D. McDysan, E. Roch, and M. Shirazipour. OpenFlow-enabled SDN and Network Functions Virtualization. *Open Network Foundation*, pages 1–12, 2014.

26. E. Keller, S. Ghorbani, M. Caesar, and J. Rexford. Live migration of an entire network (and its hosts). *Proceedings of the 11th ACM Workshop on Hot Topics in Networks - HotNets-XI*, pages 109–114, 2012.

27. C. Dixon, D. Olshefski, V. Jain, C. DeCusatis, W. Felter, J. Carter, M. Banikazemi, V. Mann, J. M. Tracey, and R. Recio. Software defined networking to support the software defined environment. *IBM Journal of Research and Development*, 58(2):1–14, 2014.

28. R. Klöti, V. Kotronis, and P. Smith. OpenFlow: A security analysis. In *Proceedings - International Conference on Network Protocols, ICNP*, pages 1–6, 2013.

29. A.G. Fallis. Security Analysis of the Open Networking Foundation (ONF) OpenFlow Switch Specification. *Journal of Chemical Information and Modeling*, 53 (9):1689–1699, 2013.

30. S. T. Ali, V. Sivaraman, A. Radford, and S. Jha. A Survey of Securing Networks Using Software Defined Networking. *IEEE Transactions On Reliability*, 64(3): 1–12, 2015.

31. D. Kreutz, F. M.V. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, page 55, 2013.

32. C. Perrin. The CIA Triad. Retrieved February 12, 2016. URL *http://www.techrepublic.com/blog/security/the-cia-triad/488*.

33. R. Meyran. DefenseFlow: The First Ever SDN Application That Programs Networks for DoS/DDoS Security. Retrieved March 10, 2016. URL *https://blog.radware.com/security/2013/04/defenseflow-dosddos-security/*.

34. P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for OpenFlow networks. *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, pages 121–126, 2012.

35. M. Mendonca, S. Seetharaman, and K. Obraczka. A flexible in-network IP anonymization service. In *IEEE International Conference on Communications*, pages 6651–6656, 2012.

36. A. Gember, C. Dragga, and A. Akella. ECOS: leveraging software-defined networks to support mobile application offloading. *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 199–210, 2012.

37. J. M. Kizza. *Guide to Computer Network Security*. Springer Science and Business Media, London, 2013.

38. M. Roesch. The SNORT Project. Retrieved January 11, 2017. URL *https://www.snort.org*.

39. P. D. Boer and M. Pels. Host-based Intrusion Detection Systems. Amsterdam University, 2005. URL *https://homepages.staff.os3.nl/~delaat/rp/2004-2005/p19/report.pdf*.

40. M. Tavallaee, N. Stakhanova, and A. A. Ghorbani. Toward credible evaluation of anomaly-based intrusion-detection methods. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 40(5):516–524, 2010.

41. H. Liao, C. Richard Lin, Y. Lin, and K. Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1): 16–24, 2013.

42. C. Morris. SANS - Information Security Resources. Retrieved January 6, 2016. URL *https://www.sans.org/security-resources/idfaq/what-do-you-do-after-you-deploy-the-intrusion-detection-system/1/20*.

43. Trend Micro Inc. About OSSEC. Retrieved July 26, 2016. URL *http://ossec.github.io/about.html*.

44. T. Xing, Z. Xiong, D. Huang, and D. Medhi. SDNIPS: Enabling Software-Defined Networking based intrusion prevention system in clouds. *Proceedings of the 10th International Conference on Network and Service Management, CNSM 2014*, pages 308–311, 2014.

45. J. O'Hara. Toward a commodity enterprise middleware. *Queue*, 5(4):48–55, 2007.

46. VFabric. Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP. Retrieved March 10, 2016. URL *http://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html*.

47. G. Marsh, A.P. Sampat, S. Potluri, and D.K. Panda. Scaling Advanced Message Queuing Protocol (AMQP) Architecture with Broker Federation and InfiniBand. Ohio State University, 2010. URL *ftp://ftp.cse.ohio-state.edu/pub/tech-report/2009/TR17.pdf*.

48. C. Robertson. Practical OSSEC. SANS Institute InfoSec Reading Room, 2011. URL *https://www.sans.org/reading-room/whitepapers/detection/practical-ossec-33699*.

49. Flowgrammable. SDN / Openflow / Message Layer. Retrieved November 14, 2016. URL *http://flowgrammable.org/sdn/openflow/message-layer/*.

50. S. Godard. Linux User's Manual: Mpstat Man Page. LinuxCommand. Retrieved December 5, 2016. URL *http://www.linuxcommand.org/man_pages/mpstat1.html*.

51. B. Hale. Estimating Log Generation for Security Information Event and Log Management. Retrieved September 15, 2016. URL *http://content.solarwinds.com/creative/pdf/Whitepapers/estimating_log_generation_white_paper.pdf*.

52. G. Lyon. Ncat Command Execution. Retrieved August 11, 2016. URL *https://nmap.org/ncat/guide/ncat-exec.html*.

53. Google. Drive Github. Retrieved September 14, 2016. URL *https://github.com/odeke-em/drive*.

54. Microsoft. Windows Dev Center: QueryPerformanceCounter Function Documentation. Retrieved November 4, 2016. URL *https://msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx*.

55. VMware. Guide to configure NTP on ESX servers. Retrieved November 23, 2016. URL *https://kb.vmware.com/kb/1003063*.

56. M. Lindquist. One-Way ANOVA. Retrieved November 11, 2016. URL *http://www.stat.columbia.edu/~martin/W2024/R3.pdf*.

57. K. Seguin. How Unreliable is UDP. Retrieved December 30, 2016. URL *http://openmymind.net/How-Unreliable-Is-UDP/*.

58. Allied Market Research. Software Defined Networking (SDN) Market is Expected to Reach 132.9 Billion by 2022. Retrieved June 1, 2016. URL *http://www.prnewswire.com/news-releases/sdn-and-nfv-market-growing-at-86-cagr-to-2020-520790081.html*.

59. J. Dix and A. Vahdat. Google's software-defined/OpenFlow backbone drives WAN links to 100% utilization. Retrieved March 13, 2016. URL *http://www.networkworld.com/article/2189197/lan-wan/google-s-software-defined-openflow-backbone-drives-wan-links-to-100--utilization.html*.

60. A. Hassidim, D. Raz, M. Segalov, and A. Shaqed. Network utilization: The flow view. In *Proceedings - IEEE INFOCOM*, pages 1429–1437, 2013.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 23-03-2017	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Aug 2015—Mar 2017

4. TITLE AND SUBTITLE Active Response Using Host-Based Intrusion Detection System and Software-Defined Networking	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Goodgion, Jonathan S., 2d Lt, USAF	**5d. PROJECT NUMBER** 17G139
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way Wright-Patterson AFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-MS-17-M-032
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

This research proposes AHNSR: Active Host-based Network Security Response by utilizing Host-based Intrusion Detection Systems (HIDS) with Software-Defined Networking (SDN) to enhance system security by allowing dynamic active response and reconstruction from a global network topology perspective. Responses include traffic redirection, host quarantining, filtering, and more. A testable SDN-controlled network is constructed with multiple hosts, OpenFlow enabled switches, and a Floodlight controller, all linked to a custom, novel interface for the Open-Source SECurity (OSSEC) HIDS framework. OSSEC is implemented in a server-agent architecture, allowing scalability and OS independence. System effectiveness is evaluated against the following factors: alert density and a selective Floodlight module response types. At the expected operational load of 500 events per second (EPS), results reveal a mean system response time of 0.5564 seconds from log generation to flow table update via Floodlights Access Control List module. Load testing further assesses performance at 10 - 10000 EPS for all tested response modules.

15. SUBJECT TERMS

SDN, IDS, OpenFlow, Software-Defined Networking, Intrusion Detection System

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Barry E. Mullins (ENG)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			
U	U	U	U	147	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-3636 x7979 Barry.Mullins@afit.edu

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18