Spring 3-2019

Evaluating the Impacts of Detecting X.509 Covert Channels

Cody Welu
Dakota State University

Evaluating the Impacts of Detecting X.509 Covert Channels

A doctoral dissertation submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Cyber Operations

March, 2019

By

Cody Welu

Dissertation Committee:

Dr. Kyle Cronin, Chair

Dr. Michael Ham, Committee

Dr. Joshua Stroschein, Committee

Dr. Crystal Pauli, Committee

DSU
DAKOTA STATE

DISSERTATION APPROVAL FORM

This dissertation is approved as a credible and independent investigation by a candidate for the Doctor of Philosophy degree and is acceptable for meeting the dissertation requirements for this degree. Acceptance of this dissertation does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department or university.

Student Name: Cody Michael Welu

Dissertation Title: Evaluating the Impacts of Detecting X.509 Covert Channels

Dissertation Chair/Co-Chair: _____ Date: 25 March 19

Committee member: Crystal Paul Date: 3.25.19

Committee member: _____ Date: 3/25/19

Committee member: Michael J. Hu Date: 03/25/2019

Original to Office of Graduate Studies and Research
Acid-free copies with written reports to library

Acknowledgments

The completion of this dissertation is a major accomplishment for me, and I am forever grateful to everyone who played a role in getting me to this point. I'll start off with my committee members: Dr. Kyle Cronin, Dr. Michael Ham, Dr. Joshua Stroschein, and Dr. Crystal Pauli. Thank you for being with me through this process, from helping me develop a topic for this study through refining the manuscript at the end. It is because of your professional expertise and friendship that I was able to complete this dissertation. To all of the faculty that have impacted me through my years as a student and faculty member at Dakota State, thank you. As a high schooler I knew I wanted to do something with computers, so thank you for introducing me to the crazy and fun world of cybersecurity. A huge thank you for Tom for guiding me through the whole way, and for making me actually want to get up for class at 7am!

I absolutely must acknowledge the support I've received from my family and friends. To my parents and sister: thank you for your unwavering support throughout this long process. Your words of support and encouragement and willingness to listen when I needed made all the difference in the world. To all my friends throughout all the years, thanks for being a distraction for me. It's easy to get wrapped up in work, but it's important to take time for yourself and get away from things for a bit, too.

Abstract

This quasi-experimental before-and-after study examined the performance impacts of detecting X.509 covert channels in the Suricata intrusion detection system. Relevant literature and previous studies surrounding covert channels and covert channel detection, X.509 certificates, and intrusion detection system performance were evaluated. This study used Jason Reaves' X.509 covert channel proof of concept code to generate malicious network traffic for detection (2018). Various detection rules for intrusion detection systems were created to aid in the detection of the X.509 covert channel. The central processing unit (CPU) and memory utilization impacts that each rule had on the intrusion detection system was studied and analyzed. Statistically significant figures found that the rules do have an impact on the performance of the system, some more than others. Finally, pathways towards future related research in creating efficient covert channel detection mechanisms were identified.

Declaration

I hereby certify that this project constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the project describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Cody Welu

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

Many aspects of our lives today use technology that relies on making connections over the internet. Most consumers will interact with online banking, social networking, online shopping, and more (Conti, Dragoni, & Lesyk, 2016). As these platforms and computer networks around the globe continue to process and store private information, the need to detect and prevent cyber-attacks is ever rising.

One technique that attackers often employ during a cyber-attack is that of a covert channel. Covert channels allow attackers to transmit data over an otherwise legitimate-looking communications channel (Zander, Armitage, & Branch, 2007). Covert channels can be used to bypass network firewalls, secretly transferring data in and out of a network. Malware has used covert channels to for command and control traffic, which allows remote attackers to interact with the malware once it has infected a computer (Dietrich et al., 2012). Additionally, malware authors have used covert channels when removing or exfiltrating data from a computer network (Carrara & Adams, 2016).

A recent instantiation of a covert channel was created using X.509 certificates by Reaves (2018). X.509 certificates are used in the transport layer security protocol to secure communications across networks (Wazan, Laborde, Chadwick, Barrere, & Benzekri, 2016). The certificates specifically play a key role in verifying the identity of entities, often verifying the identity of web servers. While X.509 certificates traditionally play a key role in verifying this trust and securing the internet, Reaves has shown they can be used for malicious covert channel purposes.

Network defenders can employ intrusion detection and prevention systems and to aid in the detection and prevention of cyber-attacks that utilize covert channels. These systems

search for known patterns in network traffic in order to identify malicious usage (Bhuyan, Bhattacharyya, & Kalita, 2014). Before these systems can be used to accurately and efficiently detect X.509 covert channel usage, network defenders must first understand how the covert channel appears in network traffic to create intrusion detection system rules.

In an effort to enhance knowledge in the industry around detecting X.509 covert channels in a highly performant manner, the purpose of this study was to observe the performance impacts detecting X.509 certificate extension misuse using an intrusion detection system. The efficiency and performance impacts of multiple different methods of performing this detection were documented and evaluated against each other. By studying these impacts, detection mechanisms can be written with knowledge of overall performance of the intrusion detection system in mind.

Background of the Problem

After successfully gaining access to a network, attackers have been able to exist in that network undetected for a median of 101 days (Mandiant, 2018). This statistic, called dwell time, details the number of days between the evidence of first compromise that an attacker is on a network to the day the compromise is detected. Globally, dwell times have ranged from less than a week to more than 2,000 days (Mandiant, 2018). Mandiant identified a common trend showing a gap in both visibility of the network and detection capabilities – organizations are unable to successfully detect a threat's presence within their own network.

Two tools that can be used to aid in the detection and prevention of breaches are intrusion detection systems (IDS) and intrusion prevention systems (IPS). An IDS is focused solely on detecting attacks or unauthorized access to the computer or network, while an IPS

takes a similar detection approach with added capabilities to block malicious traffic as soon as it is detected (Hock & Kortis, 2015). To detect malicious traffic, these systems use knowledge from prior attacks and intrusions, much like antivirus signatures (Titorenko & Frolov, 2018). Because of this, it is important to fully understand the various tactics and techniques used by attackers to bypass the IDS and IPS security controls.

Another tool that is commonly used to prevent attacks is a firewall. Firewalls provide the ability to filter traffic entering and leaving a network by blocking specific network traffic based on predefined rules (Satasiya & Raviya Rupal, 2016). Traditionally, firewalls are placed between a private internal network and a public network, commonly the internet. They inspect the traffic flowing between the two networks, making decisions on whether to allow or block the traffic based upon the source and destination of the traffic such as the source IP address, destination IP address, network protocol, source port, and destination port (Kaur, Singh, Kumar, & Ghumman, 2015).

In an effort to detect malicious content in the network application level, the capabilities of firewalls have been advancing. Next-generation firewalls have the ability to inspect the contents of network traffic deeper, beyond that of a traditional firewall. In addition to inspecting the source and destination of the traffic, next-generation firewalls also can peer into the applications that are present in the network traffic, providing additional more stringent filtering features (Neupane, Haddad, & Chen, 2018).

To bypass security controls such as firewalls, attackers need to hide their presence through information hiding techniques. Information hiding traditionally can occur by using cryptographic techniques to obscure information in transit across a computer network. Utilizing only cryptography, one could discern that two entities are in fact communicating,

but not what they are communicating about. Attackers are also concerned with staying completely anonymous so that defenders are not aware they are communicating at all (Ritchey, 2015).

In information security, the practice of information hiding can be broken down into two relevant fields: anonymity and steganography. Anonymity uses cryptographic techniques to obscure the contents of the message as well as the source and destination of the communication. Steganography is defined as hiding secrets in otherwise normal communication so that it is undetectable to a third party observer (Ritchey, 2015). If cryptography can be broken or its use is otherwise prohibited, cryptography is ineffective in anonymizing communications (Ritchey, 2015). Instead, steganography techniques can be utilized. Covert channels can be an instantiation of steganography.

Malware and malicious actors have been using covert channels as a method to enable communication and bypass network security controls like firewalls and intrusion detection systems. Covert channels are defined as the use of existing communication channels for the transfer of information through a system's security controls (Department of Defense, 1985). In practice, communication channels on computers and networks that are traditionally open for legitimate purposes can be abused and used to send custom, non-standard data through the channel. These channels can be used to hide the presence of data leaving the network, or for covert command and control communications between infected hosts (Scott, 2008).

The Transmission Control Protocol (TCP) is a reliable protocol used for communication between computers on a network (Postel, 1981). Similarly, the User Datagram Protocol (UDP) is also a protocol used for communication between computers on a network, but with minimum overhead (Postel, 1980). Each of these protocols which are still in

prevalent use today define a port number to name the ends of the communication (Reynolds & Postel, 1994). It is through these ports that most computer communication over a network occurs. The Internet Assigned Numbers Authority has assigned port numbers 0 to 1023 to specific protocols, just a subset of the 65,535 total ports available on computer systems (Reynolds & Postel, 1994).

Some common protocols and applications that have historically been used as covert channels include Internet Relay Chat (IRC), Peer-to-Peer (P2P), Hyper Text Transfer Protocol (HTTP), and Domain Name System (DNS) (Binsalleeh, Kara, Youssef, & Debbabi, 2014). Many of these protocols were optimal choices for covert channels in part because these common protocols are generally available for communication on most computers. Selecting ports and protocols that are rarely blocked by network firewalls is a key factor in the usefulness of the covert channel.

Since most computer users will have a need to browse the internet, many network firewalls today will allow general web browsing traffic outbound. Therefore, the protocols required for general web browsing could be good choices for covert channel usage. One of these common protocols is HTTPS, or HTTP Secure. HTTPS is an extension of the Hyper Text Transfer Protocol (HTTP), with the addition of encryption using the Secure Sockets Layer (SSL) or more recently Transport Layer Security (TLS) (Clark & Van Oorschot, 2013). TLS is in use by many websites across the internet to secure information as it passes between web servers and web browsers, especially sensitive information such as passwords (Scott, 2008).

TLS makes use of public key certificates, specifically X.509v3 certificates, to allow the server and client to be authenticated (Dierks & Rescorla, 2008). X.509 is the standard that

defines the format of public key certificates. X.509 certificates are a key component of a trust model used to verify the identity of entities on the internet. This is key to confirming that you are actually communicating with whom you expect, not some third party or malicious actor. A trusted third party, the certificate authority, verifies the identity of an entity and provides an X.509 certificate containing cryptographic keys to the entity (Wazan et al., 2016).

X.509 certificates are used to secure network traffic from various applications across the internet. This prevalence can make it a good candidate for use as a covert channel. In January 2018, Jason Reaves published research into using X.509 extensions as a covert channel (Reaves, 2018). The research also includes proof-of-concept code demonstrating the feasibility of this new covert channel mechanism.

Malware has been observed to make use of covert channels for communication, from the exfiltration of data to command and control traffic, instructing the malware to carry out its malicious tasks. Morto, Katusha, and Feederbot are examples of some malware families that have used DNS as a covert communication mechanism (Binsalleeh et al., 2014). DNS has been used as a covert channel, and there have been various studies done on the detection of this covert channel (Binsalleeh et al., 2014). Newer covert channels, like X.509, benefit from the existence of little to no detection mechanisms, and often will go undetected.

To modify the X.509 certificate to add arbitrary data, the attacker would need to re-create the certificate each time. This method of using X.509 as a covert channel is effective when simply using a self-signed certificate, rather than one signed by a legitimate certificate authority (Reaves, 2018). Typically, computer users are presented with a warning if the certificate is self-signed, or otherwise untrusted. However, users too often do not verify the

certificate when receiving these warnings, especially those users who encounter self-signed certificates frequently (Callegati, Cerroni, & Ramilli, 2009).

In the X.509 covert channel proof of concept, there is no need to pass the certificate to a web browser on the affected machine – the user may not know a malicious certificate is being used at all (Reaves, 2018). The certificates used in the covert channel are passed between the attacker's server and the malicious application on the victim's machine. To effectively mitigate this, an organization might choose to block all self-signed or otherwise untrusted certificates at the network level, though some organizations and users routinely encounter legitimate self-signed certificates (Callegati et al., 2009).

Statement of the problem

The problem is that X.509 certificates and X.509 certificate extensions can be misused giving end-users a false sense of security and enabling a mechanism for covert channels that enable command and control, data exfiltration, or other malicious purposes unknown to current detection mechanisms. This is evidenced by Reaves' (2018) covert channel proof of concept.

As malicious actors use new methods to circumvent security controls, there becomes a need to detect that behavior. Therefore, as X.509 misuse as a covert channel becomes more relevant and widespread, the need for effective and performant detection mechanisms will rise. As any new offensive technique is created and used, eventually will come a defensive mechanism to block or at least detect the technique in the ever-evolving cyber security cat-and-mouse game. Extensive and robust X.509 covert channel detection mechanisms do not yet exist (Reaves, 2018).

The RFC 5280 document defines the X.509 Internet Public Key Infrastructure (Cooper et al., 2008). The X.509 specification defines several certificate extensions, intended to associate additional attributes to the certificate (Cooper et al., 2008). There also is the ability to include private extensions that are not explicitly defined in the RFC (Cooper et al., 2008). Reaves's X.509 covert channel uses the Subject Key Identifier extension to transmit arbitrary data. Normally, this field is a string that contains a hash (Reaves, 2018).

While the use of the Subject Key Identifier extension field as a covert channel has been proven to be possible, there are other extensions in the X.509 specification that could theoretically be used to transmit arbitrary data through a covert channel. One would need to verify all potential fields and extensions to thoroughly detect malicious content that may exist in other fields. Today, firewalls and intrusion detection systems do not deeply inspect and verify the fields and extensions of an X.509 certificate exchange, likely because its abuse as a covert channel is not widely known (Reaves, 2018).

While accurately detecting the protocol abuse is important, so is doing so in a highly performant manner as poor performance has an effect on the system (Chan, Hammad, & Kundur, 2016). That is, obtaining a high true positive rate and a low false positive rate is a separate problem from having a significantly negative impact on the IDS's ability to inspect traffic with minimal impact to the CPU and therefore high traffic rates (Schaelicke, Slabach, Moore, & Freeland, 2003). As bandwidth requirements and usage grows on networks, so must rise the ability of an IDS to process that traffic. Intrusion detection systems that cannot process traffic fast enough will likely drop packets, increasing the chance a real intrusion will be missed (Bulajoul, James, & Pannu, 2013). This is why these systems should be as performant as possible.

Purpose of the Study

The purpose of this study was to observe the performance impacts of detecting X.509 certificate extension misuse using an intrusion detection system. The efficiency and performance impacts of multiple different methods of performing this validation will be documented and evaluated against each other. By studying these impacts, detection mechanisms can be written with knowledge of overall performance of the intrusion detection system in mind.

The performance of an intrusion detection system is defined as the rate at which events are processed (Mohammed Nazer & Lawrence Selvakumar, 2011a). The CPU and memory have a direct effect on the number of events processed by an intrusion detection system; an overloaded CPU can lead to dropped network packets and therefore fewer events processed (Hu, Asghar, & Brownlee, 2017). This study measured the CPU and memory usage of the system along with the number of CPU cycles, or ticks, that an individual detection rule consumed.

This study used X.509 covert channel network traffic generated with the proof-of-concept code described by Reaves (2018). This traffic was passed through an intrusion detection system with no specific detection rules in place and CPU and RAM utilization was measured. Then, the same traffic was passed through the same system with specific detection rules in place and the CPU and RAM utilization again was measured. These before-and-after data points suggest the selected experimental design as most appropriate.

The network traffic used in the study included X.509 certificates without covert channels, and X.509 certificates with covert channels. Since intrusion detection and prevention systems will process truly non-malicious data in a real-world environment, this

study also used non-malicious traffic that is not related to X.509 certificates or covert channels. This was done to also observe the impacts of the implemented detection mechanisms on different kinds of traffic.

Significance of the Study

The development of new methods for detecting X.509 covert channels and invalid X.509 certificates is significant in its potential impact alone. The early detection of breaches will reduce an intruder's dwell time. Dwell time is defined as the length of time from when an intruder gains access to a network to when their presence is discovered and eradicated. In 2017, the median dwell time on a network was 101 days (Mandiant, 2018).

As breaches become more significant, so does the need for early detection, bringing the dwell time closer to zero. In fact, breaches where the attacker has been able to exist undetected for a longer period of time also have a higher cost to the affected organization (Ponemon Institute, 2018). In fact, the average total cost of a data breach was $3.86 million in 2018 (Ponemon Institute, 2018). Advancing the quality and accuracy of detection mechanisms, including intrusion detection systems, can be a crucial step in bringing the median dwell time closer to zero. If undetected, the intruder could remain quietly in a network to carry out whatever goals the intruder has, from data exfiltration to even causing damage to systems.

The other major impact this study can have is the improvement of the efficiency of intrusion detection mechanisms. Inefficient signatures and other detection mechanisms built into intrusion detection systems can lead to poor performance and dropped network packets or inflated hardware requirements. As bandwidth usage on networks continues to rise, so will the

need to effectively monitor all this traffic. Large traffic flow rates coupled with numerous inefficient IDS rules could lead to an inability to monitor all packets without excessive hardware requirements (Chan et al., 2016). This study aims to determine the more efficient detection mechanisms to keep hardware requirements down and traffic processing ability up, while not sacrificing detection accuracy.

Early detection of intrusions can lead to smaller incidents and faster cleanup. This benefits organizations with computer systems, as well as those whose data is stored there. Intrusions may be inevitable but being able to detect and react as swiftly as possible is key to minimizing damage.

Nature of the Study

This study used a quantitative, quasi-experimental before-and-after study design. This design aims to discover the association of an intervention and an outcome (Harris et al., 2006). A before-and-after study design is the most appropriate for measuring a change or impact a variable has on an environment (Kumar, 2014). These studies compare measurements taken of a system before an intervention with measurements of that same system after an intervention (Kumar, 2014).

Quasi-experiments do not use randomization. In a true experimental design study, the assigning of test subjects to the control group or treatment group is completely random. In this study, the specific network traffic and intrusion detection system will not change but can be reused between control tests and treatment tests, or the tests with modified detection mechanisms. This study, therefore, lacks randomization.

In reviewing the limitations of this study design and their applicability to this research problem, it is important to note that this design can only measure the total change from the first measurement to the last. Because of this, the researcher must be aware that observed changes cannot be related to specific independent variables (Kumar, 2014). As such, the research will be conducted in a way that minimizes extraneous variables that may affect the measurement and outcome as much as possible.

Even with those limitations in mind, a quasi-experiment is the most appropriate research design for this study. There are three common quantitative experimental approaches: experimental, non-experimental and semi-experimental studies (Kumar, 2014). In an experimental study, the researcher intervening and observing a change. In a non-experimental study, the researcher observes a change and subsequently attempts to determine the cause of the change. For this study, a non-experimental design is not appropriate, since the researcher is to introduce a change to the IDS rules and subsequently observe the impacts. Semi-experimental designs include a combination of both experimental and non-experimental studies, but was not suited for this research for the same reasons a non-experimental design was not appropriate.

IDS signature and detection mechanisms were created to perform the detection of X.509 covert channels. The performance of these artifacts, specifically CPU and RAM usage metrics, are the major verification component. The study used the open source intrusion prevention and detection system, Suricata. Suricata is one of two popular open-source network intrusion detection and prevention systems in the industry, the other being Snort (Hu et al., 2017). Snort has the best market share of the two systems but has some shortcomings that Suricata addresses. For example, Snort will only process packets with a single CPU

thread, but Suricata can take advantage of multiprocessor systems to process packets with multiple threads simultaneously (Park & Ahn, 2017). Having the ability to process multiple packets silultaneously allows Suricata to process more network packets per second than a Snort system with identical hardware (White, Fitzsimmons, & Matthews, 2013a).

The environment consisted of three major components – a client computer to simulate the victim, a server computer to simulate an attacker-controlled server on the internet, and the Suricata intrusion detection system itself. The X.509 covert channel proof-of-concept code described by Reaves (2018) was loaded and verified on both of the client and server systems.

The Suricata IDS was configured with no rules as a starting control test. This represents the before test in this before-and-after study design. As rules and detection methods were designed, they were tested in the system using the same traffic as the starting control test. This represents the after test.

The research began with only looking at the Subject Key Identifier field, which is used in Reaves' proof-of-concept code. This field is only one of many different extensions present in the X.509 specification (Cooper et al., 2008). Subsequent detection mechanisms looked further beyond the one field to detect possible misuse of other available extensions. As the tests were conducted and data analyzed, small modifications were made to the detection mechanisms to evaluate the performance of the different methods, aiming for the most efficient and effective method.

Research Questions

This study's intent was to discover performance impacts of various detection mechanisms on an intrusion detection system. As a part of this, various intrusion detection

methods, signature-based and anomaly-based, were tested. The impacts of short, limited signatures and regular expressions vs. extensive, yet specific, signatures and regular expressions was explored, as an example. While new X.509 signatures are artifacts of this research, the primary goal remained to answer the following question:

> How will validating X.509 certificate trust and detecting X.509 certificate extension misuse using an intrusion detection system affect the CPU and RAM performance of the system?

To answer that question, the research measured the performance of the system with no detection mechanism present, as well as the performance of the system with the new mechanisms present. Each detection mechanism was measured separate from the others and compared after the data had been collected.

Hypothesis

Validating X.509 certificate trust and detecting X.509 certificate extension misuse using an inline intrusion detection system does affect the performance and throughput of the system.

Theoretical Framework

The theoretical framework acts as a guide for a researcher by leveraging an existing theory in the field of study (Adom & Joe, 2018). Utilizing existing research as a blueprint provides necessary structure to drive the research. A theoretical framework is well designed and accepted in the discipline (Adom & Joe, 2018).

This research uses the computational complexity theory to explain the relationship between the complexity of an IDS rule and the speed and efficiency of that rule.

Computational complexity can be traced back to Alan Turing and his Turing machine, the device that built the foundations of modern computation (Homer & Selman, 2014). Computational complexity examines the resources required to solve a problem, such as the time required (Loui, 1996). As such, this study examines the time it takes for distinctive X.509 covert channel detection mechanisms.

Several prior studies have examined the performance of an IDS as they examine network traffic using preconfigured signature-based detection rules. Prior work has studied the throughput of the IDS while changing a number of different variables: varied rulesets, varied workload with varied number and size of packets, varied system settings, and varied hardware configurations (White, Fitzsimmons, & Matthews, 2013b).

One way to measure the performance of an IDS is by measuring the packets per second (PPS) that the IDS can process. The PPS represents the dependent variable and is the major variable that is measured. The independent variables should not change without researcher interaction, which in this study are the IDS detection ruleset, the hardware the IDS is running on, and the traffic or packets being sent through the IDS.

Studies have shown that increased computing resources, like more processor cores, does not lead to more accurate detection, but does improve the packet handling capabilities of the IDS (Kabir & Hartmann, 2018). Increased processor performance has resulted in an increase of throughput the system can handle, since there are more computing resources to handle the complexity of detection (Saboor, Akhlaq, & Aslam, 2013).

Ruleset complexity also has a direct impact on the performance and throughput of the IDS. The Emerging Threats was originally created in support of a free and open source ruleset for Snort and Suricata. Now, the team curates two major sets of rules for use in these systems,

the ET-Open rules and the ET-Pro rules. The ET-Open rules are community contributed and are widely installed and used across Snort and Suricata systems. The ET-Pro rules are developed and further vetted by the Emerging Threats team, a group that creates and distributes IDS rulesets (White et al., 2013). The ET-Open rules are available for free and typically ship with Snort and Suricata, whereas the ET-Pro ruleset is a paid subscription-based ruleset. Researchers found a greater negative performance impact while using the ET-Free ruleset over the ET-Pro ruleset because the ET-Pro rules are tuned by professionals (White et al., 2013b). That is, the ET-Pro ruleset showed an increase in performance. This evidence shows a correlation between the detection rule and overall performance of the IDS.

Acronyms

The following list contains acronyms that are used frequently throughout this document. Definitions were retrieved from Ayala (2016).

ASN.1: Abstract Syntax Notation One

CPU: Central Processing Unit

DNS: Domain Name System

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

IDS: Intrusion Detection System

ICMP: Internet Control Message Protocol

IP: Internet Protocol

IPS: Intrusion Prevention System

KiB: Kibibytes

RFC: Request for Comments

SHA: Secure Hash Algorithm

TCP: Transmission Control Protocol

TLS: Transport Layer Security

UDP: User Datagram Protocol

Definitions

Abstract Syntax Notation One (ASN.1): A data notation scheme that can be used to represent numerous different data types (Kaliski, 1993). The structure of X.509 certificates are defined in ASN.1 syntax.

Censys: A search engine of data collected by scanning systems on the internet, notably X.509 certificates (Durumeric, Adrian, Mirian, Bailey, & Halderman, 2015).

Covert Channel: A communication channel used to hide the sharing of information between two entities, typically on top of another legitimate communication channel (Ritchey, 2015).

Domain Name System (DNS): An application layer protocol designed for the translating between domain names and IP addresses (Binsalleeh et al., 2014).

Hypertext Transfer Protocol: An application layer protocol designed for the transferring of hypertext webpages across the internet (Clark & Van Oorschot, 2013).

Hypertext Transfer Protocol Secure: An application layer protocol designed for the transferring of hypertext webpages across the internet over a secure channel (Clark & Van Oorschot, 2013).

Internet Control Message Protocol: A protocol used to send messages between computers such as when a service is not available. ICMP is commonly used by ping to test connectivity between networked devices (Ayala, 2016).

Intrusion Detection System: A tool used to identify security violations or network attacks by monitoring network traffic (Kabir & Hartmann, 2018).

Intrusion Prevention System: A tool used to identify and prevent security violations or network attacks by monitoring and blocking traffic. (Kabir & Hartmann, 2018)

Mpstat: A command-line tool that can be used to monitor the utilization of a system's CPU (Ahmad & Qazi, 2018).

Request for Comments: A document issued by the Internet Engineering Task Force that often describes an area of computer networking (Ayala, 2016). Within this document, the RFCs referenced describe various protocols or security related implementations.

Secure Hashing Algorithm: A family of hash functions that map a group of bits of any length to a bit string of a fixed length. Hashing algorithms are one-way, meaning it is not feasible to find and input that matches an output hash, only an output that matches an input (Ayala, 2016).

Snort: One of two predominant open source intrusion detection systems. (Brumen & Legvart, 2016)

Suricata: One of two predominant open source intrusion detection systems. (Brumen & Legvart, 2016)

Transmission Control Protocol: A transport layer protocol that enables two networked hosts to communicate with each other. TCP ensures the data was successfully delivered (Ayala, 2016).

Transport Layer Security: A protocol that provides end-to-end cryptographic communications security across computer networks. X.509 certificates are used in TLS (Ayala, 2016).

User Datagram Protocol: A transport layer protocol that enables two networked hosts to communicate with each other. UDP is a connectionless protocol and therefore does not provide a guarantee that data was successfully delivered (Ayala, 2016).

Vmstat: A command-line based tool that can be used to monitor the utilization of a system's memory (Mohammed Nazer & Lawrence Selvakumar, 2011b).

X.509: The certificates widely used today for verifying the identity of an entity, commonly another server or website (Uahhabi & Bakkali, 2017)

Assumptions

Throughout the course of this study, some assumptions were made. First, it is assumed that Suricata's built in rule profiling features provide an accurate measurement of the CPU utilization of each rule that is configured on the system. Second, it is assumed that the utilities used to monitor the overall CPU utilization and RAM utilization of the system are accurate and that any changes are the direct result of the ruleset configured in Suricata. To further assist this assumption, the environment was controlled as tightly as possible to limit any outside factors from impacting the study as much as possible. While it is assumed a highly capable CPU would have an impact on the performance, all tests were run on the same hardware configurations.

Scope, Limitations, and Delimitations

The scope of this study is to detect X.509 certificate extension misuse on an intrusion detection system while studying the impacts on the performance and throughput of the system. Rulesets were generated for the IDS to carry out the aforementioned verification of X.509 certificates, and the impact on the IDS was studied, specifically the CPU and RAM utilization. The intrusion detection system's self-reported rule profiling statistics were recorded for each ruleset tested in the IDS. Throughout the study, the researcher used the same traffic sample along with the same hardware and IDS configuration to ensure consistency between tests. The only changing independent variable is the ruleset used in the intrusion detection system.

There are a number of notable exclusions from the scope of this study. While the detection mechanisms and rulesets created and used in this study do accurately detect X.509 certificate extension misuse for covert channels, the study does not test the false positive rate of the rulesets. A false positive result is one that occurs when the IDS generates an alert for malicious activity, though the activity was not actually malicious. False positive results are an incorrect determination of maliciousness, and are therefore a factor in most metrics on a ruleset's accuracy (Chan et al., 2016).

This study also did not look at any interactions between rules in the IDS, built-in or custom. Only the rules in question were loaded and tested separately of all other rules. Additionally, the aim of the study is not to test the performance of the IDS or rules in a best-case environment, but rather to observe the performance of various rules under the same environment. Finally, this study was not looking at the performance of the system in a particularly high-bandwidth environment. These out-of-scope items can also be listed as

limitations of the study, as various factors, such as differences in traffic, large packet sizes, and high-bandwidth environments may affect the throughput and performance of the IDS.

This study utilized a virtual machine to run the Suriciata IDS with very specific controlled network traffic used for testing. This environment was purposefully segregated from outside influences to the best of the researcher's ability. This controlled virtual environment was not subject to certain influences that can have an effect on the performance of the IDS in a physical real-world environment. Some notable influences include a wider variety of network traffic with various packet sizes and purposes.

While another researcher's results may differ due to any number of variables to include the type of traffic tested, the performance of the hardware the IDS is running on, and the packet size, any correlation between IDS throughput and the detection mechanism and ruleset remains valid.

Chapter Summary

This chapter has established the profiling of the performance of the detection of X.509 extension misuse as the basis of this study. Attackers have a need to evade common network defenses and do so using information hiding methods, specifically by creating and using a covert channel (Scott, 2008). X.509 extensions are a medium that can be used as a covert channel, though detection mechanisms do not yet exist (Reaves, 2018).

As bandwidth utilization increases on computer networks, so does the need for an efficient and performant IDS. An IDS that is overloaded can drop packets, leading to the possibility of missing true-positive intrusions (Chan et al., 2016). As such, this study will

develop IDS rules for the detection of X.509 covert channels paying close attention to the CPU utilization of the rule.

This study uses a quantitative, quasi-experimental before-and-after study to discover the relationship of an intervention and outcome (Harris et al., 2006). Specifically, the mechanisms used to detect X.509 covert channels being the intervention, and the performance impact to the intrusion detection system as the outcome. This chapter discussed the framework used in the study, as well as the scope and limitations of the study.

Chapter 2 presents a literature review of the components of this study, including covert channels, X.509 certificates and attacks, intrusion detection systems and performance, and intrusion detection system rulesets. The literature review takes a look at the history and current state of the aforementioned topics, as well as a survey of previous research conducted in the detection of covert channels and other malicious network-based attacks.

Chapter 2: Literature Review

Chapter 1 provided an introduction to and overview of this dissertation while identifying the study's topic: to measure the CPU performance impacts of detecting X.509 covert channels on an IDS. The motivation and background of the problem was discussed, along with the research questions and research design.

What follows is a literature review focused on the major components in this research: covert channels and covert channel detection, intrusion detection methods, and intrusion detection system performance. Existing methods discussed in other research for covert channel detection and intrusion detection methods were applied to X.509 certificates in this dissertation.

Covert Channel Background

A covert channel has been defined as a "communication channel established contrary to the design of a system" (Carrara & Adams, 2016). Earlier, a covert channel was defined as one that was not intended to transfer information at all (Lampson, 1973). This definition has proved to be only partially correct, as covert communication channels have been hidden within portions of channels that were originally intended to transfer information, but in an originally unintended way (Binsalleeh et al., 2014). Generally, Kemmerer defined a covert channel as one that "uses entities not normally viewed as data objects to transfer information from one subject to another" (Kemmerer, 1983).

The Trusted Computer Security Evaluation Criteria (TCSEC) defines two categories of covert channels: storage channels and timing channels (MD., 1993). This dissertation is working primarily with covert storage channels. A covert storage channel involves the

modification of a storage space that two processes can both access and therefore use for communication. The selected storage channel could be an unused field in existing network communication protocols (Newman, 2007).

Some malware families abuse legitimate protocols to establish a covert communication channel for the exfiltration of data and for command and control purposes. It is important to understand existing successful uses of covert channels. Attackers have used a number of protocols for covert communication, including internet relay chat, peer-to-peer protocols, HTTP, and DNS (Binsalleeh et al., 2014).

Covert channels have been compared to steganography. In steganography, data is typically embedded into innocent looking traffic to hide the fact that the data is being transmitted (Jakobsen & Orlandi, 2016). Covert channels can utilize a legitimate network protocol as the carrier of arbitrary data, where steganography uses legitimate or normal looking audio or visual content (Zander et al., 2007).

Selection of covert channels often comes down to a few characteristics: the ability to transfer data and the availability of the channel. For example, on a clear majority of networks, HTTP, HTTPS, and DNS are open for free communication to the general internet. Most networks will allow users to browse the web, or at the very least portions of it, if filtered through a proxy server. HTTP can certainly carry custom data and is often open, making it a good communication channel. DNS is almost always open on a network, and even though it is not normally thought of as a good payload distribution channel, it can in fact be used as one. Early on, simply because of the fact intrusion detection systems were not inspecting DNS for malicious traffic, it was a great covert channel.

Malware can send arbitrary data through DNS queries and responses, although inefficiently. Often, traffic is passed using DNS TXT resource records. Only a very limited amount of data can exist in one query, creating a high rate of DNS queries and responses (Binsalleeh et al., 2014). The Feederbot malware, for example, uses TXT resource records to receive information, and specific domain requests to send information from the infected host. Command and control traffic to the malware is split into chunks, each with a maximum size of 220 bytes. The data is both encrypted and encoded before placed in the TXT record (Dietrich et al., 2012).

Covert channel creation can be especially fruitful in protocol standards that do not require or verify specific values in the packet header. Many protocols allow header extensions to carry arbitrary data that was not initially in the specification as a means to extend the usefulness of the protocol (Zander et al., 2007).

Another legitimate protocol that has been used as a covert channel is Voice Over IP (VoIP). Arbitrary data has been sent over both the control traffic and the audio channel. Data has been observed being embedded with the G.711 coded audio data. Even by injecting just a single bit into a sample of audio data to minimize audio quality loss, an attacker could transfer up to 8kb of information each second. More data can be embedded and transferred, at the risk of reducing audio quality (Scott, 2008).

Covert channels can also be established without using existing protocols. One example of this is the Fanny malware. The Fanny malware was a specific piece of malware that used USB drives to spread itself from computer to computer (GReAT, 2015). The malware utilized a covert channel to both receive commands to execute as well as to store the output of the commands, a covert channel used for both ingress and egress (Carrara & Adams, 2016). This

malware was also unique in the fact that it provided a communication channel for systems that were not connected to the internet, or air-gapped systems, by using a hidden storage volume in removable media.

When the existence of a covert channel is known, there are a few possible countermeasures, from most effective to least: eliminate the channel, limit the bandwidth, audit the channel, or document the channel (Zander et al., 2007). At times, completely removing the channel may not be realistic, especially if the channel is necessary for legitimate, critical network communication. At the very least, monitoring the usage of the channel, or blocking specific malicious packets used in the channel, is preferred.

According to Zander et al., another method for thwarting a successful covert channel is to normalize specific fields in the packets. Removing unnecessary extensions and zeroing unused bits can be an effective defense. A lot of analysis can be performed to determine if specific combinations of data are valid, according to the protocol's specification. For example, the urgent pointer in TCP traffic can be set to zero if the URG bit is not set. There are many more examples of specific data combinations that should not exist in normal, non-malicious TCP traffic.

The X.509 Standard

Secure Sockets Layer (SSL), later renamed Transport Layer Security (TLS), has been securing internet traffic for years. It was originally intended to be used to secure online financial and shopping websites, though today many other websites, services, and applications use SSL/TLS technologies for encryption (Levillain, 2012).

In order to provide additional security for network traffic, TLS verifies the identity of an entity typically by leveraging public key infrastructure (PKI). PKI is based on the trust model presented in the X.509 standard (Wazan et al., 2016).

PKI uses certificates to provide a binding between a domain name and TLS keys (Walsh, 2017). The X.509 certificate contains information to prove the ownership of the public key as well as what entity verified this ownership, among other information.

In the X.509 trust model there are three entities. The subject or server of the communication is the certificate holder, the relaying party (RP) is the client, and the certification authority (CA) is a trusted third party between the subject and RP. The CA verifies the subject is who they say they are and issue a signed certificate (Wazan et al., 2016). The RP uses the certificate signed by the CA along with their trust in the CA to verify the identity of the subject.

X.509 certificates are described as a sequence of objects using the Abstract Syntax Notation One, or ASN.1 structure (Cooper et al., 2008). The ASN.1 is flexible enough to represent a number of different datatypes to include integers and bit strings, for example (Kaliski, 1993). While X.509 certificate fields and structures are usually discussed using ASN.1, it's important to also understand the technique used to encode the data from ASN.1. The Data Encoding Rules (DER) define the unique encodings used for each ASN.1 data type (Kaliski, 1993). DER is used to encode the X.509 certificates described in ASN.1 into a binary format.

The fields of a standard X.509 certificate are described in the following list (Cooper et al., 2008).

- The **version** field is a number describing the version of the certificate. Acceptable values for this field are 1, 2, or 3.
- The **serial number** field is unique positive integer used to identify the certificate by the certificate authority. This field should handle values up to 20 octets in length.
- The **signature** field contains the algorithm identifier that is used to sign the certificate.
- The **issuer** field contains the identity of the certificate authority who issued and signed the certificate. This field contains a distinguished name (DN) as defined by the X.501 name type.
- The **validity** field describes the time range that the certificate authority has deemed the certificate valid. This field contains both the beginning date and ending date of the validity period encoded as either the UTCTime or GeneralizedTime standard. A certificate with no ending time will show the GeneralizedTime value of 99991231235959Z. For this field, values will use Greenwich Mean Time.
- The **subject** field contains the identity of the entity the certificate is issued to. This field contains a distinguished name (DN) as defined by the X.501 name type.
- The **subject public key info** field contains the public key along with the algorithm that is used.
- The **unique identifier** field or fields contain additional information to identify the certificate if subject or issuer names are reused. This is an optional field and will only appear if the certificate is version 2 or 3.

- The **extensions** field contains a sequence of one or more certificate extensions. This is an optional field and will only appear if the certificate is version 3.

X.509 certificate extensions.

Certificate extensions can be used to attach additional optional information to the certificate. While there are defined standard certificate extensions, custom extensions can be created and used as well. Entities that do not recognize an extension can either ignore it if the extension is not marked critical or reject the certificate if the unrecognized extension is marked critical (Cooper et al., 2008).

All extensions are described as an ASN.1 sequence with three components: extnID, critical, and extnValue. The extnID field is the identifier for the specific extension and is of the object identifier type. The critical field shows whether or not the extension is critical and defaults to false. The extnValue field is an octet string data type that contains the data or value that the extension itself carries. (Cooper et al., 2008)

Subject key identifier.

The subject key identifier extension contains a unique value to identify certificates that have a specific public key. The X.509 specification recommends key identifiers be created one of two ways: by computing a 160-bit SHA-1 hash of the subjectPublicKey or combining the value 0100 with the least significant 60 bits of the aforementioned SHA-1 hash. Using another means of generating a unique identifier is also standards-conforming.

This extension begins with the ID of the subject key identifier extension, which is 2.5.29.14. Encoded with DER, this value is represented as the following three bytes: `55 1d 0e`. What follows is an encapsulated octet string which contains the subject key identifier

itself. Encoded with DER, this field is first noted by the byte `04` to signify an octet string, followed by the length of the octet string field (Cooper et al., 2008).

The length of this field could be a feature used to help identify tampering or otherwise anomalous content. In Reaves' (2018) X.509 covert channel proof-of-concept code, arbitrary data was placed in this field with a maximum length of 10000 bytes. The user could, however, modify the code to only encode a certain number of bytes in each payload.

At the time of this writing, Censys, a website that utilizes internet scanning data to map the internet, had cataloged over 1.2 billion X.509 certificates with this field (Durumeric et al., 2015). Of those certificates, nearly 99.98% of them had a subject key identifier field with a length of 20 bytes. Only approximately 266,000 of those certificates, or approximately two hundredths of one percent, exhibited subject key identifier field lengths other than 20 bytes. While it can be easy to bypass, this metric is one that could be used in an intrusion detection system to detect anomalous usage of the subject key identifier field of an X.509 certificate.

Authority key identifier.

The authority key identifier extension contains a unique value to identify the specific public key corresponding to the specific private key used to sign the certificate. This may be necessary if a certificate authority has more than one signing key. The X.509 specification recommends key identifiers be created one of two ways: by computing a SHA-1 hash of the subjectPublicKey or combining the value 0100 with the least significant 60 bits of the aforementioned SHA-1 hash. Other means of generating a unique identifier is also standards-conforming.

The extension begins with the ID of the authority key identifier extension, which is 2.5.29.39. Encoded with DER, this value is represented as the following three bytes: `55 1d 23`. What follows is a sequence which contains the keyIdentifier octet string (Cooper et al., 2008).

At the time of this writing, Censys had cataloged over 1.2 billion X.509 certificates with the authority key identifier extension (Durumeric et al., 2015). Of those certificates, over 99.99% of the authority key identifier fields exhibited a length of 20 bytes. Only approximately 106,000 of those certificates exhibited authority key identifier field lengths other than 20 bytes.

Subject alternative name.

The subject alternative name extension is used to identify additional identities that are bound to the certificate. The identities listed in the subject alternative name are present as identities in addition to the subject field of the certificate. This field can contain identities of a wide variety to include a DNS name, an IP address, an email address, or a uniform resource identifier, among others (Cooper et al., 2008).

The extension begins with the ID of the subject alternative name extension, which is 2.5.29.17. Encoded with DER, this value is represented as the following three bytes: `55 1d 11`. What follows is one or more sequences which contain the specific alternative name identifiers (Cooper et al., 2008).

Key usage.

The key usage extension is used to identify the purpose of the key in the certificate. This can be used to restrict how the certificate's key is used (Cooper et al., 2008). Further,

according to Cooper et al., There are 9 bits that can be set for various options in the key usage extension. This is represented as a single byte in the certificate.

- Bit 0 is the digital signature bit. It is set when the subject public key is to be used to verify digital signatures with the exceptions of signatures on certificates and certificate revocation lists.
- Bit 1 is the non-repudiation bit. It is set when the subject public key is to be used to verify digital signatures with the same exceptions as the digital signature bit. This bit is set when the subject public key is used to provide non-repudiation.
- Bit 2 is the key encipherment bit. It is set when the subject public key is to be used for enciphering private keys.
- Bit 3 is the data encipherment bit. It is set when the subject public key is to be used for enciphering user data directly without the use of an additional symmetric cipher.
- Bit 4 is the key agreement bit. It is set when the subject public key is used for key agreement, often when Diffie-Hellman is used to manage keys.
- Bit 5 is the key certificate sign bit. It is set when the subject public key is used for the verification of signatures on a certificate.
- Bit 6 is the certificate revocation list signing bit. It is set when the subject public key is used to verify signatures on certificate revocation lists.
- Bit 7 is the encipher only bit. This bit should only be set when the key agreement bit is set. When both bits are set, the subject public key can only be used for enciphering data during a key agreement.

- Bit 8 is the decipher only bit. Similar to bit 7, this bit should only be set when the key agreement bit is set. When both bits are set, the subject public key can only be used for deciphering data during a key agreement.

The extension begins with the ID of the key usage extension, which is 2.5.29.15. Encoded with DER, this value is represented as the following three bytes: `55 1d 0f`. What follows is a single byte representing the configured usage options (Cooper et al., 2008).

Extended key usage.

The extended key usage extension defines how the public key is to be used. This field is in addition to the basic key usage field described above. This extension is typically only used in end entity certificates. There are six unique identifiers that may appear in this extension as a usage purpose (Cooper et al., 2008). These purposes are detailed in the list below of identifier number followed by the purpose.

- ID 1: Server Authentication
- ID 2: Client Authentication
- ID 3: Code Signing
- ID 4: Email Protection
- ID 8: Time stamping
- ID 9: OSCP response signing

The extension begins with the ID of the extended key usage extension, which is 2.5.29.37. Encoded with DER, this value is represented as the following three bytes: `55 1d 25`. What follows is a sequence which contains the key purpose ID (Cooper et al., 2008).

CRL distribution points.

The CRL distribution points extension defines how certificate revocation list information can be acquired. The extension begins with the ID of the CRL distribution points extension, which is 2.5.29.31. Encoded with DER, this value is represented as the following three bytes: `55 1d 1f`. What follows is a sequence which contains a DistributionPoint object. This object contains three optional fields: distributionPoint, reasons, and cRLIssuer (Cooper et al., 2008).

Basic constraints.

The basic constraints extension defines whether the subject is a certificate authority as well as the maximum number of certificates in a valid path. This allows users to determine if the public key in the certificate can be used to verify signatures. The extension begins with the ID of the basic constraints extension, which is 2.5.29.19. Encoded with DER, this value is represented as the following three bytes: `55 1d 13`. What follows is a sequence that contains a boolean value to indicate whether the key can be used for certificate verification and optionally an integer defining the path length constraint (Cooper et al., 2008).

There are many more common standard extensions defined in RFC 5280 that could be inspected and verified as conforming as a part of an intrusion detection system. The difficulty, however, is in the fact that custom extensions can be created and used legitimately.

X.509 Certificate Authority Trust Issues

One major function of X.509 certificates is to validate the identity of an entity, commonly another server (Uahhabi & Bakkali, 2017). Because of this, users must place their trust in the Certificate Authority who created and signed the certificate issued to that entity (Walsh, 2017). Certificate authorities represent a trusted third party who verifies the identity

of an entity on the user's behalf. Certificate authorities not only manage their own certificates but often build internal relationships and recommend trusting other certificate authorities on a user's behalf (Wazan et al., 2016).

Certificates bind a public signing key to an entity, a binding that is verified by the certificate authority. Often these certificates and their associated keys are used by web servers to encrypt web communications. In this case, a certificate authority often verifies the entity by verifying control over the domain, creating a domain validated (DV) certificate (Clark & Van Oorschot, 2013).

If a certificate authority does not accurately validate the identity of an entity before issuing a certificate, that certificate may be issued to an entity with a false identity (Wazan et al., 2016). This could lead to issues with users trusting the certificate authority. A system compromise or security breach is another event that could lead to trust issues with a certificate authority. In an example of a compromise, certificates could be issued by compromised certificate authorities to malicious domains (Chen et al., 2018a).

One of the more high-profile compromises of a certificate authority occurred in 2011 when the Comodo certificate authority shared that one of their registration authorities had been hacked. In the breach, attackers obtained certificates from a number of domains, some of them high-profile domains like google.com (Roosa & Schultze, 2013). Later in 2011, the DigiNotar certificate authority discovered a similar breach.

Another trust issue arises when a certificate authority issues a SubCA certificate to an organization. With a SubCA certificate, that organization can perform a man-in-the-middle attack on HTTPS browser traffic within their organization. In 2012, the Trustwave certificate

authority issued one such certificate. In 2013, the Turktrust certificate authority issued a SubCA certificate in error to the Turkish government (Roosa & Schultze, 2013).

These and other certificate authority trust issues have led to several researchers studying various methods for validating that trust, sometimes using a decentralized model. Uahhabi and Bakkali (2016) described an approach to assist the users of certificate authorities in making trust decisions. The research provides a means to determine the certificate trust level (TLoCERT) while also automating trust decisions. Factors used to determine the certificate trust level include the correctness of a certificate's content, procedures described in the certificate authority's certificate policy and certification practice statements, and the reputation of the security of a certificate authority.

Removing the need to explicitly trust one entity, that is a certificate authority, is a common theme among researchers. In one possible solution, multiple certificate authorities would be required to authorize the signing of a certificate preventing any single certificate authority from generating an unauthorized certificate (Chen et al., 2018b). Another study suggests implementing a witness cosigning model that requires a large decentralized group of witnesses to verify certificates signed by a certificate authority (Syta et al., 2016). Again, requiring many entities to verify a certificate reduces the possibility of an unauthorized certificate being created and signed.

Pinning describes a process where the client can pin a known key to a certificate on the first visit to a site. When the user visits the site again and is presented with the certificate, the pinned key should exist. If the pinned key does not exist, it is possible a man-in-the-middle attack is taking place. This method works similar to how OpenSSH servers and clients operate (Vikan, 2015).

While the trust issues with the model have been widely publicized and solutions researched, no real changes to the widely-used model have been made. Introducing a change to such a widely-used system can be difficult and costly.

X.509 Covert Channels

The X.509 standard defines the format of public key certificates, commonly used in TLS/SSL for securing web traffic across the internet. Each time a client web browser visits a website secured with TLS/SSL, a handshake will occur, and the certificate details will be exchanged using the X.509 standard. A vast majority of networks will leave port 443 and HTTPS open to the internet to allow normal web browsing. This wide availability of the protocol and usage of X.509 certificates make it a great choice for a covert channel on almost any network.

The second feature a protocol should possess to be used as a covert channel is to have space or fields to store, transmit, and receive arbitrary attacker-controlled data. While there are plenty of fields and extensions in the typical X.509 certificate, one of the X.509 standard's security features could largely prevent the tampering of any data. A digital signature can be computed based on the contents of the certificate, and compared to the signature encrypted with the certificate authority's public key (Scott, 2008). If the computed digital signature does not match the signature that was encrypted with the certificate authority's public key, one would know the contents of the certificate had been tampered with. If one was attempting to communicate through an existing legitimate program like a web browser, this would be an effective defense. However, in the case the attacker has control of the affected system with a custom application, like malware, the attacker will not worry about verifying the contents of

the certificate. The attacker would only be concerned with evading common network perimeter defenses such as intrusion detection systems.

We can assume if the attacker is attempting to create a covert channel to either exfiltrate information or setup command and control communications, they would already have control of the system. Placing a custom software application on the system to interact via abused X.509 channels would be trivial. Additionally, using self-signed certificates, an attacker could generate a certificate with secret data. Carlos Scott (2008) reminds us the intent of using covert channels is "not to hide the data being exfiltrated, but to hide the fact that the transmission is taking place, making it appear as regular traffic" (p. 32).

As an attacker is embedding arbitrary data into the X.509 certificate for transfer, the attacker must pay attention to all of the fields in the certificate. Simple network traffic analysis has the ability to detect suspicious values in the certificate like invalid or otherwise odd validity dates on the certificate. To evade certain defensive mechanisms, the data placed in the fake certificate must look as though it is valid or otherwise normal data for the certificate (Scott, 2008).

Carlos Scott (2008) identified three fields in the X.509 specification that have the potential to be used to transfer arbitrary data covertly.

- The **serialNumber** field in a certificate is used to uniquely identify a certificate. A unique positive integer generated by the issuing certificate authority, the field's potential wide range of legitimate values could be used as a covert channel. An attacker would need to encode the hidden data to fit the positive integer requirement.

- The **validity** field contains two dates to document when the validity period for the certificate begins, and when the period expires. Scott (2008) suggests encoding data as the difference of the two dates. Valid dates are presented as either UTCTime or GeneralizedTime and can be between the year 1900 and the year 9999. There are 10,642,086,000 seconds within that timeframe, each of which could be used to encode a unique value.
- The **unique identifiers** fields contain a bit string of arbitrary length. Because of this openness of the specification, it is easier to place arbitrary data in these fields and still follow the X.509 standard. The limitation of these fields, though, is they are deprecated.

Another location an attacker may be able to embed arbitrary data in the X.509 certificate format is through the use of the certificate extensions. These extensions allow for the association of additional data to the certificate. There are fifteen standard extensions defined in RFC 5280. The specification also allows for the creation of additional custom extensions if defined by both parties of the communication, though recommends against it for compatibility reasons (Cooper et al., 2008).

One of the defined standard X.509 extensions is the SubjectKeyIdentifier. This field is intended to identify certificates that contain a specific public key. The RFC states, "Applications are not required to verify that key identifiers match when performing certification path validation" (Cooper et al., 2008). The field should normally contain a hex string. Reaves made use of this field to send data from an attacker-controlled server to a client by encrypting arbitrary data and converting it to a hex string before placing it in this field

(Reaves, 2018). To an IDS, this field that should be a hex string, is still a hex string – potentially bypassing some detection methods.

Transport Layer Security

Transport Layer Security (TLS) is a protocol that is widely used to implement data integrity, authentication, and confidentiality across the internet (Rescorla, 2018). TLS was preceded by the Secure Sockets Layer (SSL) protocol which was originally created over 20 years ago (Albashear, Ali, & Ali, 2018). TLS was originally detailed in 1999 as RFC 2246 to succeed SSL (Dierks & Allen, 1999). The most recent revision of the TLS protocol, version 1.3, was outlined in RFC 8446 (Rescorla, 2018).

In a TLS channel, the server side of the communication will always be authenticated while the client can be authenticated as an option. TLS provides confidentiality, that is the data communications over the secure channel can only be visible to the intended parties of the communication. Finally, TLS provides for the detection of the tampering of data sent over the channel, therefore providing data integrity (Rescorla, 2018).

The two major components of TLS are the handshake protocol and the record protocol. The handshake protocol takes care of the authentication and negotiation of cryptographic methods and keys to be used during the secure communication. The record protocol leverages the agreed upon cryptographic methods and keys to protect data as it passes through the secure channel (Rescorla, 2018).

In the first part of the handshake protocol the client initiates the connection with a client_hello message. The server then responds back to the client with a server_hello message. During this first phase as a part of the hello messages, the client and server determine the security and encryption capabilities of each endpoint (Scott, 2008).

In the second part of the handshake protocol the server sends a certificate message to the client TLS utilizes X.509 certificates for authentication of the server, specifically the X.509v3 specification outlined in RFC 5280 (Rescorla, 2018). X.509 certificates are the most widely utilized method by websites to authenticate themselves to establish trust with users (Fu, Li, Xiong, Cao, & Kang, 2018).

In the third phase of the handshake protocol the client verifies the validity of the X.509 certificate the server provided during the second phase. After verification, a client_key_exchange message is sent to the server which is then verified by the server in the fourth and final phase of the handshake protocol (Scott, 2008). The record protocol will then begin with the encryption and transfer of records, or data.

X.509 Man-in-the-Middle Attacks

Man-in-the-middle (MITM) attacks are one of the more well-known attacks of major concern to computer security professionals (Conti et al., 2016). This technique occurs when the attacker is positioned between the user and the server and can intercept communications. When the attacker sees the initial communication from the client to the server, the Client Hello, the attacker replies with an illegitimate, forged certificate to the client, establishing communication with the victim client (Conti et al., 2016). The client may believe they are communicating with the intended server but in reality, are communicating directly with the attacker.

By stealthily taking control of the communications between a client and server, the man in the middle can view or change the traffic passing between the client and server (Pingle, Mairaj, & Javaid, 2018). Inserting oneself into a communication stream, the

attacker's view of the traffic is exactly as it passes through the network. This traffic may be in plain text, but transport layer security using X.509 certificates is often used to provide encryption to this communication.

To view the traffic passed during an encrypted session, the MITM attacker will have to remove the encryption. One method supported by the open source tool Ettercap is to hijack the SSL/TLS session itself. In this attack, the user is interacting with the attacker, and the attacker is interacting with the website. Instead of presenting the user with the legitimate website's certificate to setup encrypted communication, the attacker presents the user's web browser with a certificate the attacker owns to setup encryption between the user and attacker, often a self-signed certificate. In this case, the traffic is encrypted while in transit, but the attacker is able to decrypt the traffic before sending it along to the legitimate web server (Pingle et al., 2018). Another similar attack uses a tool called SSL-strip to remove the X.509 certificate and never setup an SSL/TLS session between the user and web server, but only between the attacker and web server.

MITM attacks using X.509 certificates do not exist only using self-signed, attacker-generated certificates. One such counterexample is a compelled certificate creation attack. In this attack, government agencies may obtain a trusted certificate from a legitimate certificate authority by compelling that certificate authority to create one (Soghoian & Stamm, 2012). This false certificate could be used to covertly intercept traffic that is otherwise thought to be secure using X.509 certificates.

Another MITM attack scenario puts the certificate authority directly in the spotlight. A browser will determine a presented certificate is valid if it can trace back it's signing to a certificate authority that the browser trusts (Walsh, 2017). It is possible, however, that a

trusted certificate authority either be coerced or become compromised and generate a forged certificate (Dacosta, Ahamad, & Traynor, 2012). This example of a MITM attack is also stealthy, as the user and browser may not know the otherwise trusted certificate was forged by a trusted authority. As an example, a forged certificate was used to intercept nearly 300,000 Gmail sessions in Iran (Leyden, 2011). Additionally, the certificate authority Trustwave admitted to supplying technology to a third party, allowing them to issue SSL certificates (Leyden, 2012). These and other incidents degrade the implicit trust that users and browsers place in certificate authorities.

Intrusion Detection and Prevention Systems

Two tools that can be used to aid in the detection and prevention of breaches are intrusion detection systems (IDS) and intrusion prevention systems (IPS). An IDS is focused solely on detecting attacks or unauthorized access to the computer or network, while an IPS takes a similar detection approach, with added capabilities to block malicious traffic. To detect malicious traffic, these systems use knowledge from prior attacks and intrusions, much like antivirus signatures (Titorenko & Frolov, 2018).

These systems can be broken down further into two additional categories: host-based and network-based. The network-based systems place their focus on monitoring traffic as it flows across a network, inspecting the contents of the network packet itself along with the header information, like the source and destination of the traffic. Host-based systems focus on monitoring a specific computer, inspecting log files, file system integrity, and other malicious activity on the kernel of the computer and the computer as a whole (Bhuyan et al., 2014; Warzynski & Kolaczek, 2018).

Each of these types of systems use one or both of two typical detection strategies: signature-based and anomaly-based. Signature-based systems detect threats through the use of present knowledge of existing threats and attacks. Signature-based systems utilize preconfigured rules to look for known indicators of compromise. This method is extremely effective at detecting known attacks, but cannot detect new attacks (Warzynski & Kolaczek, 2018). The searching for known patterns in traffic has also been referred to as misuse-based intrusion detection (Bhuyan et al., 2014).

Anomaly-based systems look for a deviation from normalcy. For anomaly-based methods to be successful, it must first be possible to understand what normal actually looks like. Researchers have applied both statistical methods to determining if an event's occurrence is an anomaly as well as machine learning algorithms (Warzynski & Kolaczek, 2018). Anomaly-based detection methods have the ability to detect new or previously unknown attacks, but they may also exhibit a higher false-positive rate than signature-based detection mechanisms.

When discussing the accuracy of a system's detection methods, four different detection accuracy rates are studied (Bhuyan et al., 2014).

- The **true positive rate** refers to the number of malicious events that are correctly classified as malicious divided by the total number of actual malicious events. This is also referred to as the hit rate.
- The **false positive rate** refers to the number of events that are incorrectly classified as malicious divided by the total number of non-malicious events.

- The **true negative rate** refers to the number of events that are correctly classified as normal or non-malicious divided by the total number of non-malicious events.
- The **false negative rate** refers to the number of events that are incorrectly classified as negative divided by the total number of actual malicious events.

When comparing the performance of signature-based and anomaly-based detection mechanisms, signature-based detection mechanisms have a very high true positive rate and very low false positive rate. To the contrary, anomaly-based detection mechanisms often can have a high false positive rate. Generating many false alarms requires additional human intervention and manual analysis to determine if an alarm is actually a true positive or false positive (Warzynski & Kolaczek, 2018).

In a typical setting for a network-based intrusion detection or prevention system, the system is placed on a network's connection to the internet. In this configuration, the system is able to monitor all traffic passing between the specific network being monitored and the entire internet (Bhuyan et al., 2014). Host-based intrusion detection systems are placed on the hosts themselves, providing a more complete view of the activities occurring on a network. Networks are different, however, and exact placement of the system on a network depends on elements like the budget and the environment (Bashir & Chachoo, 2014).

Many different network-based intrusion detection systems are deployed on networks today including commercial and open-source solutions. Three popular open-source solutions are Snort, Bro, and Suricata. All three of these tools monitor network traffic and decodes it to match against pre-defined rules, but they are each implemented a bit differently. Snort uses a single thread to perform signature detection in an effort to minimize hardware requirements.

Suricata is quite similar to Snort, though it utilizes multi-threading to perform signature detection in a more performant way. Unlike the two previously mentioned tools, Bro's features are geared more towards classifying network traffic. Bro's classified network traffic is commonly used for anomaly detection and behavior analysis (Hu et al., 2017). Both Snort and Suricata can be used in either intrusion detection system or intrusion prevention system mode.

Intrusion Detection and Prevention System Rules

While intrusion detection systems (IDS) and intrusion prevention systems (IPS) may vary slightly, the basic structure of their signature-matching rules will be quite similar. This section will explore the different components that make up an IDS rule, as well as different rule-based intrusion detection methods.

Intrusion detection systems use rules to define tests that are carried out on each packet of network traffic that it inspects to perform signature matching. If any of the rules configured in the IDS contain a match to the packet in question, an alert is raised to indicate the match and therefore the possibility of an attack. As the industry learns of more and more attacks and vulnerabilities, the number of rules that are created to detect these also increases at the same rate (Afzal & Lindskog, 2015).

All IDS/IPS rules will have a few things in common: the action of the rule, protocol, source and destination addresses, source and destination ports, and additional rule options or keywords. The specific options or keywords tend to vary a bit more between intrusion detection systems.

IPS rules can be quite broad, possibly only looking for traffic sent from a specific IP address, or to a specific port. Rules can also get more complex, as they may add any number of features that detect specific keywords deep within a packet's payload.

Intrusion detection signatures or rules typically will analyze many components of an IP packet. Shaelicke, et al. explain the two categories of rules in a typical IPS, based on what sections of the packet they analyze:

> "Header rules inspect the packet header in an attempt to detect specific combinations of features, such as the source and destination address, port numbers, checksums or sequence numbers. Payload rules attempt to match a specific byte sequence in a packet's payload." (Schaelicke et al., 2003, p. 3)

Most IDS and IPSs utilize a flexible language to define rules that detect certain activity and carry out a corresponding action. The rule language used by Snort and Suricata places each rule on its own row with two logical segments: the rule header and the rule body (Kuang, Mei, & Bian, 2012). The rule header constitutes of the following five key components:

- The **action** refers to what the system will do if a match is found, such as generating an *alert* or to *drop* the matched packets.
- The **protocol** field defines the network protocol used that must be present for a match, such as *tcp*, *udp*, *icmp,* or *ip*. The *ip* protocol is used for any protocol. In Suricata, there are also a number of application layer protocols available, such as *http, tls,* and *dns* among others.

- The **source** field is a combination of the IP address and port that must be the source of the packet for a match. This field may also contain the *any* value which will match all source address and ports.
- The **destination** field is a combination of the IP address and port that must be the destination of the packet for a match. This field may also contain the *any* value which will match all destination address and ports.
- The **direction** field defines the direction of the packet between the source and destination fields. This field is typically filled with -> which matches the traffic from the source to the destination. The other option for this field is <> which would match the rule in both directions between the source and destination.

The rule body portion of the Snort and Suricata rules contains any of a number of options to define the contents of a packet. Basic firewalls can prevent traffic from flowing between specific IP addresses over specific ports, but it is the additional content and detection mechanisms that make an IDS or IPS more extensible than a traditional firewall. Suricata refers to the rule body portion as the rule options. The most common keyword in the rule options is the content keyword which allows the rule author to define specific characters or bytes that must be present in the packet for a match. There are a number of modifiers that can define where in the packet that specific content should be checked for. As an example, a rule author can look for the content “index.php” specifically in the http_uri portion of the HTTP packet (OSIF, 2016).

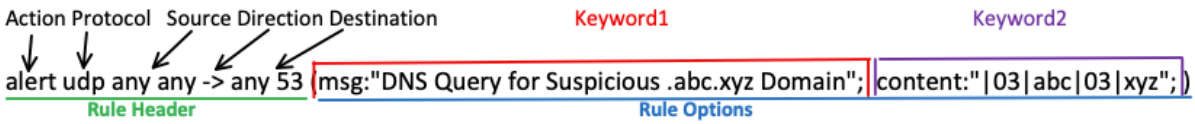

Figure 1 Suricata Rule Structure

Figure 1 shows the typical structure of a Suricata rule starting with the rule header followed by the rule options. The header of this rule begins with the action, in this case alert. Following the action is the protocol field. This rule will match only packets using the UDP protocol. Next is the IP address and port source and destination fields. This rule will match packets with a destination port of 53, no matter the source port or IP addresses. The next section of this rule contains two rule options. The first is a message keyword which specifies a message that will accompany the alert that this rule generates in the Suricata log files. Finally, the last rule option is a content keyword. This rule will only match packets that have the byte `03` followed by the text `abc`, followed by the byte `03`, followed by the byte `xyz`. If all of the content information is found in the packet and the packet matches the options specified in the rule header, an alert will be generated.

Measuring Intrusion Detection and Prevention System Performance

Typically, the effectiveness of an intrusion detection system and its signatures is measured by comparing the true-positive and false-positive rate the signature produces. Another metric that is less often measured is the impact on the throughput of an IDS that a signature has. With the prevalence of high-bandwidth environments ever increasing, any new detection mechanisms or rules in an IPS must have as minimal of an impact to the bandwidth processing capabilities of the system.

The performance of an IDS is affected by many different potential factors: hardware resource availability, IDS rule design, and the type and bandwidth of network traffic. Network traffic with a large number of small packets will have a significant impact on header based rules, whereas smaller number of large packets will have a similar impact on payload based rules (Schaelicke et al., 2003). Real network traffic, of course, is comprised of various different packets and packet sizes, so an IDS needs to handle all kinds of potential network traffic. As such, this research suggests measuring the performance of an IDS through using different packet payload sizes, ranging from the tens of bytes to thousands of bytes, to test the range of all possible packet sizes.

Some studies have pitted different intrusion detection systems against each other, comparing their performance and resource usage. In one such study, performance measurements of Snort and Suricata were compared. Researchers measured the CPU and memory load on the server after one minute of processing network traffic, as well as the number of dropped packets (Brumen & Legvart, 2016).

Another study used constant bandwidth and a varying number of IPS rules to measure the performance. Once packet loss is observed, the IPS is considered to be overwhelmed and unable to process additional rules or any additional bandwidth. Schaelicke et al. (2003) suggest “the total number of rules that a platform is able to support is a measure if the platform’s NIDS capabilities” (p. 4).

It has been proven that a larger quantity of packets in a constant time, and a constant number of packets in a shorter time, thus higher bandwidth in both cases, can lead to dropped packets by an IPS. Researchers have also found that, all other factors equal, larger packet sizes can lead to an increase in dropped packets in an IPS (Bulajoul et al., 2013).

Another study has shown a relationship between the number of rules configured in the IDS and the time it takes for the IDS to inspect a single packet. As the number of rules increase, so does the amount of processing time. In the study, the time to process a packet against 10 rules was 4.2 milliseconds, while the time to process the same packet against 1000 rules was 15.4 seconds. On the high end, 1.5 million rules took 563.8 milliseconds (Kuang et al., 2012).

The Snort and Suricata intrusion detection systems include a rule profiling feature that can be useful in analyzing a specific rule's performance. This is typically used to find inefficient rules that may plague the throughput performance of the IDS. The IDS's keep track of the number of clock cycles, or ticks, a specific rule consumes. Statistics reported include the total number of ticks consumed, maximum for one check, and average per check, among others. This feature will aid in the performance analysis of different signature detection methods.

It seems most existing research in the area of intrusion detection system performance analysis, outside of accuracy, has centered around discovering and measuring factors that have the greatest impacts on IPS throughput, mainly network traffic characteristics and underlying resource availability. Not much academic research has focused on looking at the performance of the IDS rule designs themselves and the impact they may have on the throughput of the system. This dissertation proposes to perform that study – specifically with X.509 covert channel detection rules.

Chapter 3: Research Methodology

The purpose of this study is to examine the performance impacts of various rules and methods of detecting X.509 covert channels on an intrusion detection system. This chapter presents the research methodology used to accomplish the purpose of this study. Details on the research method and design used will be presented, followed by the procedures and instrumentation used for data collection, processing, and analysis.

Research Method and Design Appropriateness

This study used a quantitative, quasi-experimental before-and-after study design. The goal of this study was to measure and compare the performance impacts of various rules and methods of detecting X.509 covert channels on an intrusion detection system. Measuring the performance change of the intrusion detection system after introducing new rules being the key goal, the quasi-experimental before-and-after study is the most appropriate choice. The quasi-experimental before-and-after research design aims to discover the association of an intervention and an outcome (Harris et al., 2006).

The two primary research designs are qualitative and quantitative research. Qualitative research is typically used to explore social and human events and in order to understand subjective experiences. Qualitative studies are also categorized as those with open-ended questions rather than close-ended questions, or studies that often use words as a key data point rather than numbers. Additionally, qualitative studies feature researchers observing a setting, rather than collecting data from instrumentation (Creswell & Creswell, 2018).

Quantitative research is typically used to measure and test relationships between variables to explain or evaluate interactions between the variables (Leavy, 2017). Quantitative

research is often characterized by numbers and figures collected from instrumentation, unlike qualitative studies. Additionally, quantitative research is used to test theories by examining relationships among variables (Creswell & Creswell, 2018).

If the focus of a study is to gather information on beliefs, understandings, and similar data points, a qualitative study design would be appropriate. However, studies that aim to obtain specific measurements of variations, a quantitative study design is more appropriate (Kumar, 2014). As the aim of this research was to study the relationship between two variables, a quantitative research design was most appropriate to specifically measure and evaluate the relationship between an intrusion detection system rule and the performance of the system.

Two forms of quantitative research design are survey design and experimental design (Creswell & Creswell, 2018). Survey design research provides descriptions of trends or tests the association between variables within a population. Unlike experimental research, the researcher does not manipulate variables during the research, and rather only observes and documents phenomena. In experimental design, the researcher manipulates a variable and observes the impact that manipulation has on the outcome of other variables (Creswell & Creswell, 2018). In experimental research, an action is performed before observing the impact of that action (Leavy, 2017). The effects of that action are isolated to be the result only of that action by holding other variables constant (Creswell & Creswell, 2018).

Similar to experimental design is a casual-comparative design. In casual-comparative, the researcher attempts to find relationships between the independent and dependent variables, searching for which variable had the affect being studied (Salkind, 2010). Casual-comparative research is also known as ex post facto research. Similar to finding relationships,

correlational research design measures two variables to determine how they are related or to find patterns between the two variables (Privitera, 2013). Correlational research aims to determine to what extent two variables are related, not to determine the extent of an effect one variable has on another.

In true experimental design, the subjects of the research are randomly placed into groups, like the control group and the experimental group. The control group does not experience the action or intervention during the study but rather serves as a reference point when comparing the experimental group. If the researcher were to not use a control group, they would not be able to determine the significance of an effect on the population, since no group remains constant and unaffected throughout the research. The experimental group does receive the action or intervention during the study and can be compared to the control group in the study.

Unlike a true experimental design, quasi-experimental designs do not use randomization when assigning test subjects to groups. One common type of quasi-experiment is the time-series experiment. In this experiment, measurements are taken from a single group for a period of time without intervention. Then, the group is given the experimental intervention and measured again for the same period of time (Leavy, 2017). In this study, the variables in the control and experimental group are exactly the same, being the network traffic and intrusion detection system. This was ensured to be true by utilizing the exact same network traffic and test system as a part of a tightly-controlled virtual environment to provide consistency in variables between tests. Virtualization allows the same computer system to be copied and reused between multiple control and treatment tests, and the study therefore lacks randomization. This is why a quasi-experimental research design is best for this study.

This study used a quantitative, quasi-experimental before-and-after design. This design aims to discover the association of an intervention and an outcome (Harris et al., 2006). During this study, the researcher made an intervention through modifying the rules configured in the IDS and observed any performance change in the system as a result. A before-and-after study design is the most appropriate for measuring a change or impact a variable has on an environment (Kumar, 2014). These studies compare measurements taken of a system before an intervention with measurements of that same system after an intervention (Kumar, 2014). This study looks specifically at the impact that specific rule designs have on the performance of an intrusion detection system when detecting X.509 covert channels.

In reviewing the limitations of this study design and their applicability to this research problem, it is important to note that this design can only measure the total change from the first measurement to the last. Because of this, the researcher must be aware that observed changes cannot be related to specific independent variables (Kumar, 2014). As such, the research will be conducted in a way that minimizes extraneous variables that may affect the measurement and outcome as much as possible.

Research Question, Hypothesis, and Variables

The research question that guided this study was: *How will validating X.509 certificate trust and detecting X.509 certificate extension misuse using an intrusion detection system affect the CPU and RAM performance of the system?* To answer that question, the research measured the performance of the system with no detection mechanism present, as well as the performance of the system with the new mechanisms present. Each detection mechanism was measured separate from the others and compared after the data had been collected.

The hypothesis for this study was: *Validating X.509 certificate trust and detecting X.509 certificate extension misuse using an inline intrusion detection system does affect the performance and throughput of the system.* This question and hypothesis guided this study's data collection and analysis methods, along with the specific variables that were observed. The specific variables measured were the CPU and RAM utilization of the system along with the number of CPU cycles the processing of each rule consumed.

Population

This study's population was comprised of an exclusively virtual machine environment. Virtual machines allow users to set up one or more operating systems on one physical computer (Ansari, Hans, & Khatri, 2017a). In enterprise-scale environments, virtual machines allow for the maximum possible utilization of physical resources by placing more than one virtual machine and workload on the hardware. Virtual machines were used for this study due to their ability to be easily replicated for subsequent tests, and ability to be operated in a completely controlled separate environment.

Virtual machines operate on a software layer called a hypervisor. The hypervisor takes care of the allocation of computing resources to the virtual machine or virtual machines running on the hypervisor. Hypervisors are typically classified in two different ways: type I and type II. Type I hypervisors are software that operates directly on the physical hardware, where a type II hypervisor software runs on top of another host operating system (Ansari, Hans, & Khatri, 2017b). This study used a Type I hypervisor, specifically VMware ESXi, to minimize any additional processing overhead that would be caused by the underlying host operating system in an environment with a type II hypervisor. The virtual machines operating

on the type I hypervisor were limited to only the machines necessary for this study to better control and limit any influences or performance impacts from outside systems.

At the core of the test environment is the intrusion detection and prevention system. Today, there are two major players in the open source intrusion detection system market. Prior to the introduction of Suricata in 2009 by the Open Information Security Foundation, the open source Snort intrusion detection system was the industry standard signature-based intrusion detection system (Albin & Rowe, 2012). Suricata is very similar to Snort but was designed to process network traffic faster by employing multi-threaded processing. Multi-threaded applications can more effectively take advantage of modern multi-core central processing units, which provides the Suricata system the ability to process more network traffic than a single-threaded application (Albin & Rowe, 2012).

In this study, the Suricata system was installed on CentOS 7.6 operating system, which was the most recent version of CentOS at the time. CentOS was chosen due to its popularity in enterprise environments, close relationship with the Red Hat Enterprise Linux distribution, and prior use in similar studies (Albin & Rowe, 2012; Brumen & Legvart, 2016; Khamphakdee, Benjamas, & Saiyod, 2014; Shah & Issac, 2018). Additionally, CentOS is a free Linux distribution, making replication of this study easier. The system was configured with one virtual CPU at 2.4GHz and 4GB RAM, though the specific hardware resources such as quantity of RAM or CPU configured is not important in this study, since the results were not intended to be compared across other systems. The aim of this study is to compare the performance of various rules against each other on the same hardware.

While the Suricata intrusion detection system is the primary focus system of this study, consistent network traffic for the system to observe needed to be created, specifically

traffic that exhibited Reaves' (2018) X.509 covert channel design. On the aforementioned hypervisor, two Windows 10 systems were set up to simulate the client and server necessary to run the X.509 covert channel. Each of the systems were configured with 2 virtual CPUs at 2.4GHz each and 4GB RAM. The existing proof-of-concept code was loaded and ran on each system to transmit custom information over X.509 certificates, specifically the SubjectKeyIdentifier extension. The Suricata system was able to inspect the traffic between the systems by receiving a mirror of all the traffic on the virtual network switch that the devices were connected to as illustrated in Figure 2.

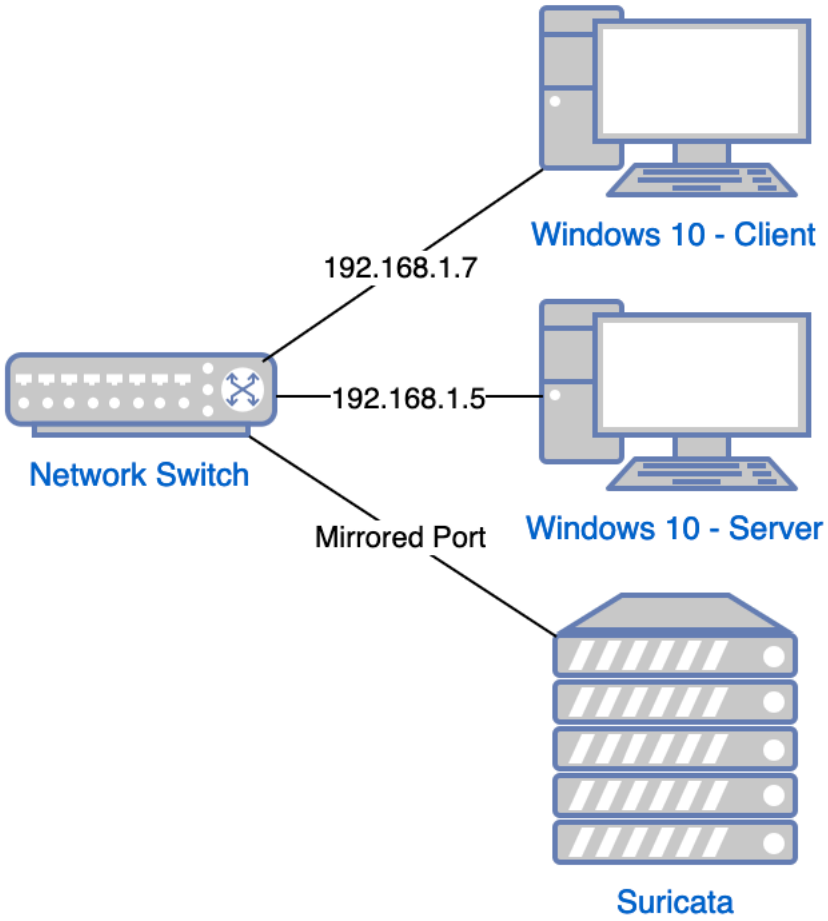

Figure 2 Suricata Intrusion Detection System and Windows Client and Server with X.509 Covert Channel

As information was transferred over the X.509 covert channel between the Windows 10 server and the Windows 10 client computers, the Suricata system received a copy of the traffic for inspection. Since there were multiple tests as a part of this before-and-after study, tests would need to be run against the exact same network traffic. With this in mind, traffic was captured on the Suricata machine while the X.509 covert channel was being used between the Windows machines. The traffic was captured with the tcpdump utility on the Suricata machine. Tcpdump is one of the most popular packet-sniffing tools available and has the ability to capture, display, and write out the data that is being transmitted over an ethernet network (Fuentes & Kar, 2005). In the case that tcpdump may have not accurately captured all traffic, the same captured file was used for all tests and does not affect the validity of the study (Arlos & Fiedler, 2016).

Capturing the traffic once and using it repeatedly for all tests ensures consistency between tests. Some computers, like the Windows machines in this environment, may periodically generate additional network traffic such as checking for updates or telemetry information. By capturing the traffic once and removing any unnecessary systems from the environment, the network traffic that Suricata will analyze between tests is consistent, leading to more accurate results.

Research Design

The purpose of this study was to observe the performance impacts of detecting X.509 certificate extension misuse using an inline intrusion detection system. Previous research by Reaves (2018) has shown X.509 certificate extensions, specifically the SubjectKeyIdentifier extension. The two key components of this study are to measure and observe the performance

of the system, specifically the impacts on the CPU and RAM of the system, and to create rules for the system to detect X.509 extension misuse.

The performance of the system is of utmost importance as network throughput rises and intrusion detection and prevention systems need to inspect a larger volume of traffic just as quickly as before. This study looks specifically at how the complexity of rules has an impact on the performance of the system. The before and after nature of this study looks at exactly that – the performance of the system before and after a new detection method or rule was introduced to the system. The major factor in determining how quickly a system can process traffic is to look at how long it takes the CPU of the system to process the traffic. As such, the number of operations a CPU must carry out to analyze a single packet of traffic is a major metric in this study.

Prior to creating or testing any rules or detection mechanisms, a capture of network traffic was first obtained of the proof-of-concept X.509 covert channel. Rather than re-creating the covert channel communications for every test, a static packet capture file provided the consistency required for the researcher to control all possible variables to give the most accurate test results. If a static capture was not used, there is the possibility that other network features on the systems in the network can introduce various unwanted network traffic into the environment. One such example is that of a computer checking the internet for updates, an action that may not be consistently present or otherwise if a static file is not used for the tests.

The Suricata system was configured with no rules as a starting control test. This represents the before test in this before-and-after study design. As rules and detection methods

were designed, they were tested in the system using the same network traffic capture as the starting control test. This represents the after tests.

This study began with only looking at the Subject Key Identifier extension, which is used in Reaves' proof-of-concept code. This field is only one of many different extensions present in the X.509 specification (Cooper et al., 2008). Subsequent detection mechanisms were explored beyond the one field to detect possible misuse of other available extensions. As the tests were conducted and data analyzed, small modifications were made to the detection mechanisms to evaluate the performance of the different methods, aiming for the most efficient and effective method. Short, limited signatures and regular expressions vs. extensive, yet specific, signatures and regular expressions were explored and tested, as well as anomaly-based detection mechanisms.

By gathering RAM and CPU performance metrics of the system both before and after the detection mechanisms were deployed, the research can clearly show the significance of the impact of the detection mechanism on the overall performance of the intrusion prevention system, or lack thereof. The measurements can be used to tune the methods, so they are as performant as possible without sacrificing true-positive detection rates. It is important to keep in mind, though, that a high true-positive detection rate was not the primary objective of this study.

Sampling Frame

In quantitative studies, the researcher attempts to select a sample of the population being studied in a way that the sample can represent the population as a whole (Kumar, 2014). Selecting a sample of the population may be necessary in studies where the entire population cannot reasonably be a part of the study. While selecting samples of a population for a study

may be necessary based on time and resource constraints, it is not without drawbacks. The researcher is only able to use the sample to estimate the entire population's reaction to the study (Kumar, 2014). Still, appropriately selected samples can produce reasonably accurate results.

Utilizing all possible variants of network traffic including all possible protocols and various packet lengths and structures for the tests in this study was not feasible. Therefore, a representative sample was selected. Non-random sampling is used when the size of a population is not known (Kumar, 2014). As the whole population of possible network traffic packet contents is not known, non-random sampling was used. In some non-random sampling scenarios, the researcher may utilize their own judgement as to what sample of the population can provide the best sample to fulfil the objectives of the study. This is called purposive sampling (Kumar, 2014).

To answer the research question of this study, a sample that includes traffic exhibiting Reaves' X.509 covert channel is necessary. In order to describe the possible impacts of detecting this traffic on a larger population of network traffic on the Internet, additional traffic needs to be selected as a part of the sample. Network traffic that was captured during a cyber defense competition was selected due to its wide variety of traffic types. Additionally, this network traffic made available for future research was not generated for a specific purpose, but rather was a capture of various types of traffic on a real network.

Data Collection

The main metrics collected during this study were the CPU and RAM utilization of the intrusion detection system. These performance statistics were collected on the system as a whole during the inspection of traffic, as well as by Suricata as it relates to the CPU clock

cycles, or ticks, of each rule configured on the system. The CPU and memory usage have a direct effect on the number of events processed by an intrusion prevention system; an overloaded CPU can lead to dropped network packets and therefore fewer events processed (Hu et al., 2017). Rules that demonstrate a higher processing impact would then theoretically reduce the number of packets or bandwidth a system can inspect than a system with a more efficient rule, all other computing resources being equal.

During each of the tests conducted, data was collected from instrumentation described in the next section of this chapter. A few specific metrics were collected from the tools to determine the CPU and RAM utilization of the system while processing network traffic. When collecting the CPU utilization, the specific percent of CPU utilization was noted in one second intervals during the testing. Each of the values over the timeframe of an individual test were used to calculate the mean value of CPU utilization for one specific test. A similar method was conducted using the number of ticks consumed to check one packet against an IDS rule as well as the quantity of random access memory consumed by the system in kibibytes (KiB).

It should be noted that additional system resources may be consumed by introducing these various performance tracking solutions. However, the design of this study is to analyze the performance of various rules on the same system, not across separate systems or hardware. Therefore, because these performance trackers are enabled during the testing of each rule, the theoretical performance impacts are the same across all tests.

Instrumentation

Tools built into many Linux-based operating systems were leveraged to accurately measure the CPU and memory utilization of the system. Additionally, rule profiling built in to Suricata was used to measure specific rule performance details.

One utility used in this research is the mpstat utility. This measures the CPU utilization of a system as often as the user would like, up to as frequently as once every second. Since the time it takes to process all packets in a given test is much longer than one second, sampling every second was sufficient to capture the utilization while packets were being processed. In this study, mpstat was used to measure the CPU utilization of the system while Suricata was processing and inspecting the network traffic. The tool shows a number of CPU utilization statistics, two of which were essential to this study: % user time and % system time. The % user time shows the percentage of CPU utilization for executing commands at the user or application level, and the % system time shows the percentage of CPU utilization for executing commands at the system or kernel level. Mpstat details CPU utilization percentages to the hundredth of a percent.

The utility used in this research to measure the memory utilization of the system is the vmstat utility. Built in to CentOS, this utility operates much like the mpstat utility, measuring system performance as often as the user would like, up to as frequently as once every second. Since the time it takes to process all packets in a given test is much longer than one second, sampling every second was sufficient to capture the utilization while packets were being processed While vmstat can be used to measure statistics on the utilization of the CPU, memory, and input/output devices, this study used the tool primarily to collect information about memory utilization. By default, mpstat gives more detailed CPU utilization statistics than vmstat. Both mpstat and vmstat are widely known tools to measure resource utilization

on Linux systems, and have been employed by researchers on many previous studies (Ahmad & Qazi, 2018; Casalicchio & Perciballi, 2017; Tesfatsion, Klein, & Tordsson, 2018).

The Suricata intrusion detection and prevention system can profile and report on the performance of the individual rules configured on the system. This is the main source of performance data on individual rules collected during this study. It is important to note that Suricata must be compiled with rule profiling options enabled as an option. While Suricata can generate reporting on a number of rules configured on the system at the same time, for this study only one rule will be configured and tested at a time to remove any interactions between rules that may affect the performance counters of any individual rule.

Suricata rule profiling generates five key data points used in this study: ticks, checks, matches, max ticks, and average ticks. A tick refers to a single CPU clock tick and is therefore the number of CPU clock cycles spent on analyzing network traffic against a single rule. Checks refers to the number of times the system checked a network packet for a match against a specific rule. Matches is simply the number of network packets that a rule was found to match. Max ticks and average ticks refer to the highest number of CPU clock cycles spent analyzing one network packet and the average number of CPU clock cycles spent on each network packet analyzed, respectively.

Validity and Reliability

Two measures of quality in a quantitative study are validity and reliability. Validity is the concept of accurate and correct measurements in the study, and reliability refers to the accuracy of the measuring instrumentation (Heale & Twycross, 2015). As this study primarily leveraged performance information through CPU and RAM utilization data, this section will look at the validity and reliability of the instrumentation used to collect such data.

This study used multiple tools to collect CPU and RAM utilization data on the system: mpstat, vmstat, and the internal Suricata rule profiling features. Using multiple data collection methods in the study helps to verify the validity of the data that each method produces. Data that is not consistent between multiple instruments is a threat to the validity of the study (Onwuegbuzie, 2000a). All three of these data collection utilities can generate CPU utilization metrics and their results can be compared to ensure the accuracy and therefore internal validity of the measurements gathered. Additionally, the same instrumentation was used for all tests throughout the study, preventing additional instrumentation threat to internal validity (Creswell & Creswell, 2018).

In many studies, population validity is a major threat to external validity. Population validity refers to how the results of the research on a small or subset population may be extrapolated to cover a larger population (Onwuegbuzie, 2000b). To discuss the population validity of this study, the Suricata intrusion detection system utilized as well as the network traffic used to test the rules should be presented. The exact results and data points of this study will likely not match up with external systems as the underlying hardware, such as the CPU, has different capabilities on different systems and will therefore perform differently. The relative test results when comparing rule treatments, however, should hold on other systems. Additionally, the specific results such as the number of CPU clock cycles spent on an individual rule and network packet will vary based on the contents of that network packet. While it is difficult to obtain a sample set of network traffic more indicative of all environments, the relative test results when comparing rule treatments should still hold on other systems.

To demonstrate the reliability of the instrumentation used in the study, subsequent tests, all variables equal, should continually produce consistent and stable results (Kumar, 2014). Two methods that can be used to verify instrumentation reliability are the test/retest procedure as well as running parallel forms of the same test. The test/retest procedure employs the instrumentation during a test and notes the results. After, the same test is run under the same or similar conditions, and the results are noted once more. The results of the two test are compared, and the same or similar results suggest the instrumentation to be reliable (Kumar, 2014). The test/retest method was selected over running parallel studies so that the same instrumentation and environment could be utilized between tests.

Additionally, by running multiple tests, this study aims to prevent any observational bias. Observational bias is when an insufficient sampling of the data has been collected, often when the phenomena being studied is not observed for a prolonged period of time (Onwuegbuzie, 2000b). Observational bias can be a threat to the validity of the study. Since this study conducted the same test repeatedly over a period of time, observational bias is not a threat.

Data Analysis

The data generated by the instrumentation described previously was collected from raw text files and gathered into a spreadsheet. The data then undergone editing, reducing, and analysis as recommended by Kumar (2014) to prepare the data for further statistical analysis. The primary goal of the editing and reducing phases was to organize the data in a spreadsheet in categories based on the data type, and to remove any data collected before and after the system began processing network traffic.

The data collected during the study was then interpreted and presented by using both inferential and descriptive statistics. Descriptive statistics provide a method to summarize and describe the data (Leavy, 2017). Two of the three kinds of descriptive statistics were used, those being measures of central tendency and measures of dispersion (Leavy, 2017). Measures of central tendency determine a single value to represent the sample, like mean, median, and mode values. In this study specifically, the median values were calculated and presented across test iterations of the same rule or lack thereof. Measures of dispersion provides a method to show the magnitude to which the individual items differ from each other. Standard deviation is the measure of dispersion that shows how individual values relate to all of the values within a set (Leavy, 2017). In this study, the standard deviation of the results was calculated and presented. The descriptive statistics were also presented visually using tables and bar charts. The third variation of descriptive statistics, frequencies, was not appropriate to describe the results of this study.

Inferential statistics allow inferences to be made about the entire population (Leavy, 2017). The null hypothesis was tested using null hypothesis significance testing, or statistical significance tests. While there are more than two groups of results, one for each of the rules tested, the t-test was chosen to compare the results of just two groups at a time in this before-and-after study.

Summary

The purpose of this study was to examine the performance impacts of various rules and methods of detecting X.509 covert channels on an intrusion detection system. This chapter presented various research methodologies, detailing the specific research methodology selected to accomplish the purpose of this study. Details on the research method

design as they relate to this study were also discussed. Additionally, the specific instrumentation used for data collection was presented along with data processing, analysis, and statistical analysis techniques were reviewed and discussed. The next chapter presents the data that was collected as a part of this study as well as a detailed statistical analysis of the results of this study.

Chapter 4: Results

The purpose of this study was to examine the performance impacts of various rules and methods of detecting X.509 covert channels on an intrusion detection system. Chapter 3 outlined the research methodology and system design to accomplish the purpose of this study. Statistics were gathered on the CPU and RAM utilization of the system as well as metrics on the amount of CPU time each rule in the intrusion detection system consumed. This chapter presents the collected data and results in terms of answering the research question presented in chapter 1: How will validating X.509 certificate trust and detecting X.509 certificate extension misuse using an intrusion detection system affect the CPU and RAM performance of the system?

Data Collection

The focal point in the environment for this study is the Suricata intrusion detection and prevention system, as described in Chapter 3. To ensure consistency, the same static packet capture was used in each test. The packet capture used contained X.509 covert channel traffic along with various other network traffic that an intrusion detection system may process on most networks.

During a single test, the Suricata system was started along with the vmstat and mpstat CPU and RAM utilization profiling tools. Vmstat and mpstat each kept a record of current resource utilization on the system every second. Subsequently, the packet capture was played back from an external system using tcpreplay across Suricata's network interface. Once the packet capture was fully replayed, vmstat, mpstat, and Suricata were all shut down and the performance data generated during the test was saved in text files. Repeated tests were

conducted with all independent variables held equal to provide more accurate results. A total of 10 iterations of each test was conducted for this study, the same as in similar tests of intrusion detection systems when researchers have conducted repeated tests 10 times to provide accurate statistics (Hu et al., 2017).

Each test of 10 iterations was conducted with a different rule or rule set configured in Suricata. Most tests were conducted with just one rule configured to give a better comparison of the performance impact each individual rule has on the system. Each of the rules verified various components of the X.509 certificate using different mechanisms. Each is described in detail in the results section in this chapter.

After the tests were run and the data from each test was saved in separate text files, all of the data was gathered and loaded into a spreadsheet for further reduction and analysis. Each of the three types of data were loaded into separate spreadsheets: CPU utilization, RAM utilization, and Suricata rule performance data. Statistical analysis was then performed on the data which is presented in the following sections.

Results

After all tests were conducted, the data was collected and organized. Data collected using vmstat and mpstat was trimmed to remove data points collected prior to and after Suricata was processing network traffic. The tests that were conducted were grouped into categories based on the contents of the Suricata rules. The sections that follow first present details on the rules being tested followed by descriptive statistics and statistical tests (Leavy, 2017).

Suricata Rules

A total of 16 different Suricata rules were tested in this study to detect the existence X.509 covert channels. Three specific X.509 extensions were chosen as candidates for the study, as their contents could be routinely verified. As explained in more detail in Chapter 2, other extensions in the X.509 protocol have varied contents which makes them difficult to statically verify with an intrusion detection system rule. The three fields selected are the subject key identifier, authority key identifier, and key usage identifier. Each of these fields have common features among a vast majority of certificates surveyed in the Censys certificate repository such as length and how they are computed (Durumeric et al., 2015). Each of the rules used in this study are available in Appendix A. Table 1 provides a brief description of each of the rules created and tested during this study.

Table 1. *Generated Rules*

Rule ID	Description
1010	3-byte subject key identifier exists - TCP
1011	3-byte subject key identifier exists - TLS
1020	3-byte subject key identifier exists, and 4-byte length is incorrect - TCP
1021	3-byte subject key identifier exists, and 4-byte length is incorrect - TLS
1030	3-byte authority key identifier exists - TCP
1031	3-byte authority key identifier exists – TLS
1040	3-byte authority key identifier exists, and 4-byte length is incorrect – TCP

1041	3-byte authority key identifier exists, and 4-byte length is incorrect – TCP
1050	3-byte key usage exists – TCP
1051	3-byte key usage exists – TLS
1060	3-byte key usage exists, 4-byte length is incorrect – TCP
1061	3-byte key usage exists, 4-byte length is incorrect – TLS
1070	All 3-byte identifiers do not exist
1071	All 3-byte identifiers exist, but 4-byte lengths are incorrect
1072	All 3-byte identifiers exist, but 4-byte lengths are incorrect – with distance modifier
1080	Custom lua script
1082	Custom lua script with 3-byte subject key identifier

Single Extension Rules

The rules created for each of the different extensions have similar features and can be categorized together for this test. First, basic rules were created to simply detect the presence of the specific X.509 extension in the certificate. Since each of these extensions has a very specific 3-byte identifier, this made up the content of the rules. The subject key identifier rules searched for `55 1d 0e` in the certificate, the authority key identifier rules searched for `55 1d 23` in the certificate, and the key usage identifier rules searched for `55 1d 0f` in the certificate.

Each of these extensions have specific content lengths that are commonly used as outlined in Chapter 2. If the extension was being used but with a content length that differed

from what is normal for the extension, the certificate could be marked as suspicious and an alert generated. This is a method of anomaly detection. To perform this detection, rules were created that looked for the existence of the extension as described in the previous paragraph as well as did not exhibit the expected length. In all cases, the ASN structure bytes can be used to determine the length of the field. Three rules were created that looked for the presence of a three-byte sequence and looked for a four-byte length sequence. If the three-byte identifier sequence exists but the four-byte length sequence does not, an alert was generated in the intrusion detection system.

Tls and tcp rules.

Rules were created for each of the previously mentioned single exteention rules both using Suricata's TCP protocol identifier and a duplicate rule using Suricata's TLS protocol identifier. Suricata includes four basic protocols that can be matched in a rule: TCP, UDP, ICMP, and IP. Some application layer protocols are also available including TLS, HTTP, and DNS, among others (OSIF, 2016). X.509 certificates will fall under both TCP and TLS protocols. TLS is an application layer protocol that is layered on top of TCP, a reliable transport protocol (Dierks & Rescorla, 2008).

Multiple extension rules.

The rules described in the previous section were created individually for each extension – that is, any one rule would only detect the existence of a single X.509 extension. All of these rules were combined into a single larger rule that checks for all of the features described above. This rule searches for and matches on the existence of three different three-

byte extension identifiers, and the lack of three different four-byte extension length descriptors. This rule looks for the aforementioned content in any location in the entire contents of the packet.

This rule was then duplicated and further modified with distance keywords. The distance keyword in a Suricata rule describes a relationship between the previous two content keywords. This limits the area in the packet payload that Suricata searches for the second content keyword based on the first content keyword (OSIF, 2016). In this rule, a distance of 0 was specified for each ID and length content pairs. This meant that the four-byte length descriptors must be found after the three-byte extension identifier in the contents of the packet.

Lua scripting.

Lua is a programming language that Suricata rules are able to leverage for additional functionality when necessary. At times, the content matching features of a Suricata rule are not sufficient to perform accurate detection. The entire packet or payload being inspected can be sent to a lua script's match function where additional computation or algorithms can be executed (OSIF, 2016). The lua script can be written to contain additional code to further inspect the contents of the network packet in question.

To perform additional verification on the packet contents, tools outside of basic Suricata rules may need to be employed. In this case, additional verification of the contents of the subject key identifier extension requires tools beyond that of the basic Suricata rule. The data of the subject key identifier extension is normally created by computing the SHA1 hash of the subject public key field in the X.509 certificate (Cooper et al., 2008). There is not a

built-in method to compute a hash of specific fields in a certificate from within a basic Suricata rule.

To perform additional verification of the subject key identifier extension, a custom lua script was created. The script has two major components: locating contents and computing a hash. First, the script locates the subject public key and the subject key identifier extension in the payload of the packet. Next, the script computes the SHA1 hash of the subject public key. Finally, the subject key identifier and the computed SHA1 hash are compared. If the contents match, it is assumed this field of the certificate is following the X.509 specification and an alert is not generated. If the data does not match, an alert is generated. The full lua script is included in Appendix B.

CPU Utilization

The CPU utilization data was gathered with the mpstat utility. Metrics on CPU utilization were taken every one second during the tests. Mpstat shows a number of CPU utilization statistics, 2 of which were essential to this study: % user time and % system time. The % user time shows the percentage of CPU utilization for executing commands at the user or application level, and the % system time shows the percentage of CPU utilization for executing commands at the system or kernel level. Combined, the % user time and system time shows the overall CPU utilization. All CPU utilization data was averaged over each of the 10 tests. Table 2 shows a description of the CPU utilization over each test, beginning with the test with Suricaca configured with no rules.

Table 2. *CPU Utilization*

Rule ID	Mean (%)	Standard Deviation

No Rules	27.6236	2.9923
1010	30.4146	2.7286
1011	28.9806	1.1932
1020	30.4996	2.6162
1021	30.112	2.4015
1030	29.0348	1.8645
1031	29.1704	1.7524
1040	28.5676	1.5757
1041	29.0866	1.9582
1050	29.0112	1.1106
1051	29.768	1.2298
1060	28.2474	1.3781
1061	29.5156	2.0205
1070	30.383	2.4867
1071	29.8928	2.0841
1072	30.6368	2.0930
1080	67.7278	1.6565
1082	29.3386	1.2130

When Suricata was configured with no rules, the overall average CPU utilization was 25.69%. This number represents a baseline, or the before value in this before-and-after study. With individual rules configured, the CPU utilization ranged from 28.25% on the low end to 68.52% on the high end. The 3-byte detection rules ranged from 28.98% to 30.41%, and the 3-byte and 4-byte detection rules ranged from 28.25% to 30.50%. CPU utilization under the

combined detection rules ranged from 29.89% to 30.64%. The most expensive rule containing the custom lua script lead to 67.73% CPU utilization.

Rule Profiling

Another method for measuring the CPU utilization of each rule is by using Suricata's built-in rule profiling features. The system keeps metrics on the total number of CPU clock cycles a rule consumes, reported as ticks, the number of times a rule was checked, and the average number of ticks per rule check. A higher number of average ticks per check indicates the rule has a more significant impact on the processor. Table 3 shows a description of the average number of ticks per check for each rule tested.

Table 3. *Average Ticks Per Check*

Rule ID	Average Ticks Per Check	Standard Deviation
1010	4179.734	519.2998
1011	3754.797	262.3186
1020	4772.063	526.8026
1021	4590.442	484.6253
1030	4617.814	412.2188
1031	4501.713	341.9629
1040	4694.486	275.3798
1041	4547.258	311.9209
1050	3836.006	213.4287
1051	3878.009	233.7582
1060	4652.633	280.2987

1061	4764.856	352.2671
1070	9365.476	871.6503
1071	5601.855	641.7245
1072	5671.773	518.2514
1080	8122695314	132412.4226
1082	8846.291	795.3751549

With individual rules configured, the average number of ticks for a rule ranged from 3754 on the low end to 8122695314 on the high end. The 3-byte detection rules ranged from 3754 to 4617, and the 3-byte and 4-byte detection rules ranged from 4547 to 4772. The average number of ticks for the combined detection rules ranged from 5601 to 9365. The most expensive rule containing the custom lua script used 8122695314 ticks.

Each of the rules above were also configured with either either the TLS or TCP protocol. When comparing the TLS and TCP rules, the TLS rules had a mean value of 4339 ticks per check, a low of 3754, and a high of 4764. The TCP rules had a mean value of 4458 ticks per check, a low of 3836, and a high of 4772.

RAM Utilization

The RAM utilization data was gathered with the vmstat utility. Metrics on RAM utilization were taken every one second during the tests. Vmstat monitors free memory in kibibytes (KiB) by default, or 1024 bytes. All memory utilization values are presented in KiB. Since Suricata was shut down between each of the ten iterations of a single test, the difference in free memory from the beginning of one iteration to the end was calculated and averaged

between all ten iterations of a test. Table 4 describes the memory consumed over the course of each 10-iteration test.

Table 4. *RAM Utilization*

Rule ID	Memory Consumed (KiB)	Standard Deviation
No Rules	7877	519.517468
1010	7536.4	429.69552
1011	7965.2	504.069995
1020	7447.2	1090.52178
1021	7474	1016.71668
1030	7244.8	939.207836
1031	7560.4	817.577788
1040	7540.4	725.749158
1041	7477.6	672.407495
1050	7361.2	840.833967
1051	7534	1233.58243
1060	7727.2	384.478036
1061	7378.4	714.808394
1070	7741.2	1033.07239
1071	7394.4	1165.31225
1072	7180	930.935014
1080	9250.4	787.83186
1082	7821.2	776.084635

With individual rules configured, the average memory utilization ranged from 7180 KiB on the low end to 9250.4.2 KiB on the high end. The 3-byte detection rules ranged from 7244.8 KiB to 7965.2 KiB, and the 3-byte and 4-byte detection rules ranged from 7378.4 KiB to 7741.2 KiB. The average memory utilization for the combined detection rules ranged from 7180 to 7741.2. The most expensive rule containing the custom lua script used 9250.4 KiB of memory.

Statistical Significance

Statistical significance tests are used in research to test the null hypothesis (Leavy, 2017). The null hypothesis is one that states there is no relationship between variables in the study. The null hypothesis in this study is: Validating X.509 certificate trust and detecting X.509 certificate extension misuse using an inline intrusion detection system does not affect the performance and throughput of the system.

There are many statistical tests that can be conducted to test the null hypothesis. Examples include an analysis of variance (ANOVA), Cramer's V, and Pearson product-moment correlation (Leavy, 2017). Cramer's V is used to determine the level of a relationship between variables. The Pearson product-moment correlation test is used to test both the strength and direction of a relationship between two variables. This study was not concerned with testing the strength of a relationship, but rather just showing that a relationship may or may not exist. An ANOVA test is used to compare the results from more than two groups (Leavy, 2017). ANOVA was considered for this test since there are many different rules that were created representing different groups. However, the null hypothesis is strictly concerned with comparing the performance and throughput of the intrusion detection system without rules configured and with rules configured. Therefore, the testing of the null hypothesis can

be satisfied by directly comparing just two groups: performance metrics with no rules configured, and performance metrics with rules configured.

A statistical test that was designed to compare the results of two research groups and is fitting for this study is the t-test (Leavy, 2017). The t-test was performed to evaluate the statistical significance of the data collected, specifically the CPU utilization data. The rule performance data was not used since there was no data collected when the system had no rules configured. The t-test was also used to calculate the standard deviation and population mean of each of the datasets. A 95% confidence level was chosen as the goal, giving an alpha *p*-value of less than 0.05 to affirmatively show statistical significance. This alpha level was chosen due to its popularity in prior research (Keppel & Wickens, 2004). Results with a *p*-value less than 0.05 shows that the data is statistically significant and can disprove the null hypothesis.

A t-test was performed on the CPU utilization data collected with mpstat. The CPU utilization of the system with no rules configured was tested against the CPU utilization of the system with the treatment rule to detect an incorrect length of the subject key identifier extension. This computation did show statistical significance with a t-value of 2.17073 and a *p*-value of 0.021786, less than the alpha *p*-value of 0.05.

When conducting a t-test on the CPU utilization of the system with no rules configured against the CPU utilization of the system with the treatment rule with a custom lua script to detect an invalid subject key identifier, the results were also statistically significant. The test showed a t-value of 35.17655 and a *p*-value of 0.00001, much less than the alpha *p*-value of 0.05.

Table 5 shows the t-value and *p*-value results of a t-test conducted between the baseline CPU Utilization when the system was configured with no rules and the CPU Utilization when the system had a rule configured. Nine of the seventeen rules tested showed statistical significance against the test with no rules configured.

Table 5. *Statistical Significance T-Test, CPU Utilization*

Rule ID	**t-value**	***p*-value**	**Significant?***
1010	2.06761	0.026681	Yes
1011	1.26372	0.111227	No
1020	2.17073	0.021786	Yes
1021	1.94567	0.03374	Yes
1030	1.2008	0.122696	No
1031	1.33818	0.098749	Yes
1040	0.83742	0.20667	No
1041	1.22732	0.117756	No
1050	1.30422	0.104295	No
1051	1.98851	0.031089	Yes
1060	0.56787	0.288569	No
1061	1.57205	0.066676	No
1070	2.1277	0.023719	Yes
1071	1.86685	0.039149	Yes
1072	2.47547	0.011737	Yes
1080	35.17655	< 0.00001	Yes

1082	1.59345	0.064233	No

* p-value < 0.05

Summary

This chapter presented the quantitative results of this study. To start the study, tests were conducted on the Suricata system when there were no rules configured. CPU and RAM utilization metrics were gathered. Subsequently, the same tests were conducted on the Suricata system with various rules configured. The goal of this study was to determine the performance impact that various detection rules have on an intrusion detection system when detecting X.509 covert channels.

The tools used for data collection during the test were mpstat, vmstat, and Suricata's built in rule profiling. Each of these tools saved the generated data in a file for later analysis. After the tests were conducted, the data was gathered from all tests and cleaned as recommended by Kumar (2014). Data points collected before and after Suricata was started and analyzing network traffic were discarded, and the data between tests was further combined and normalized.

A descriptive analysis of the data was presented, showing the mean resource utilization values on the Suricata system under different rule configurations. The results generally showed that CPU utilization on the system was increased when Suricata had rules configured versus when the system did not have any rules configured. Some of the configured rules had a greater impact on the CPU utilization than others. The rule that used a lua script to detect an invalid subject key identifier had a far greater impact on the CPU utilization than all other rules.

Finally, a statistical analysis was performed between data collected when the system was configured with no rules and when the system was configured with the rule that detects the existence of the subject key identifier extension. A t-test was conducted with a goal of a 95% confidence level. The calculated *p*-value showed that the results of the study were statistically significant.

Chapter 5: Conclusions

The purpose of this quantitative before-and-after study was to examine the performance impacts of various rules and methods of detecting X.509 covert channels on an intrusion detection system. This chapter includes a discussion of the findings of the study, comparing the performance impacts of the various rules that were created. Also in this chapter is a discussion of the limitations of the study and recommendations for future research, followed by a brief summary.

This chapter contains an interpretation of the findings as they relate to answering the research question: How will validating X.509 certificate trust and detecting X.509 certificate extension misuse using an intrusion detection system affect the CPU and RAM performance of the system?

Limitations

This study employed a quasi-experimental before-and-after design. The change in performance measurements can be described from the before test when the Suricata system had no rules configured to the after test when the Suricata system had rules for detectin X.509 covert channels configured. While the total change can be measured, the observed changes cannot be related to specific independent variables (Kumar, 2014). In a quasi-experimental design, conclusive results cannot be obtained due to the non-random sampling of the population, unlike true experiments. In this study, the environment and instrumentation used in all tests was tightly controlled to minimize any possible differences in the population in an effort to limit any impacts of non-random sampling.

Three specific extensions became the focus of the detection rules created and tested during this study. While mechanisms for fully verifying the subject key identifier field were implemented, the rules created as a part of this study are not all-encompassing in detecting all X.509 covert channels.

Network traffic that did include X.509 covert channels was used in each of the tests. Additional traffic that contained legitimate X.509 certificates and other types of network traffic was also used. The type of network traffic an intrusion detection system processes can have an impact on the performance of the system, including memory and CPU utilization (Hu et al., 2017). Different traffic flows experienced in different environments may produce different results than what was presented in this study. Additionally, this study did not use all possible variations and combinations of network packet contents during the study. A selected network packet capture was used, which may not be indicative of traffic occurring on the Internet as a whole.

It is not possible to entirely limit outside variables from affecting the testing environment and therefore the results of the study. Known variables were controlled and accounted for to best limit any outside affects. A standalone virtual environment was used to host all of the systems that are a part of the test including the Suricata IDS. This allowed for repeated testing in the same virtual machine under the same environment, though any additional outside variables that may be introduced by operating in a hypervisor may have been present. There are a number of necessary supporting processes running on the systems, both the hypervisor and the IDS, that could have an effect on the performance of the system. It is possible that outside uncontrolled variables may have had an effect on the data and results of this study.

Additionally, the any human error or omission by the researcher may have had an effect on this study. While it is assumed that the researcher had the necessary knowledge and skills to complete a quasi-experimental before-and-after study of this nature, limited experience in research design and research experience in general may lead to an impact on the study.

Discussion of Findings

Previous research has suggested that X.509 certificates can be used for the transfer of arbitrary information as a covert channel mechanism (Scott, 2008). If an attacker controls both endpoints of the communication channel, certificates could be modified on the fly to transfer data. In fact, Jason Reaves (2018) had generated a proof-of-concept design to do exactly that. Reaves' code used the subject key identifier extension of the X.509 certificate standard to transfer data as a covert channel.

This study both created rules for the Suricata intrusion detection system to accurately detect this covert channel and verify portions of the X.509 certificate, but also evaluated performance impacts of those rules. Specifically, the CPU utilization, RAM utilization, and number of CPU cycles each rule consumes upon being checked was studied. The detailed findings presented in Chapter 3 show that performing X.509 covert channel detection in this way does have a performance impact on the intrusion detection system.

As network traffic flows continue to grow in bandwidth, being able to process a high volume of traffic with an intrusion detection system is important. Previous studies have shown that increased computing resources, like more processor cores, does improve the packet handling capabilities of the intrusion detection system(Kabir & Hartmann, 2018). Increased processor performance had resulted in an increase of throughput the system can

handle (Saboor et al., 2013). Therefore, this study measured the CPU utilization that each rule caused on the system, since it is directly linked to the overall traffic processing capabilities of the intrusion detection system.

Overall CPU Utilization

To answer the main question of this study, the CPU utilization of the Suricata system was monitored both when the system had rules configured to detect X.509 covert channels, and when it did not. This allows for the measurement of the impact to the CPU of the rules, which also has an effect on the throughput capabilities of the system.

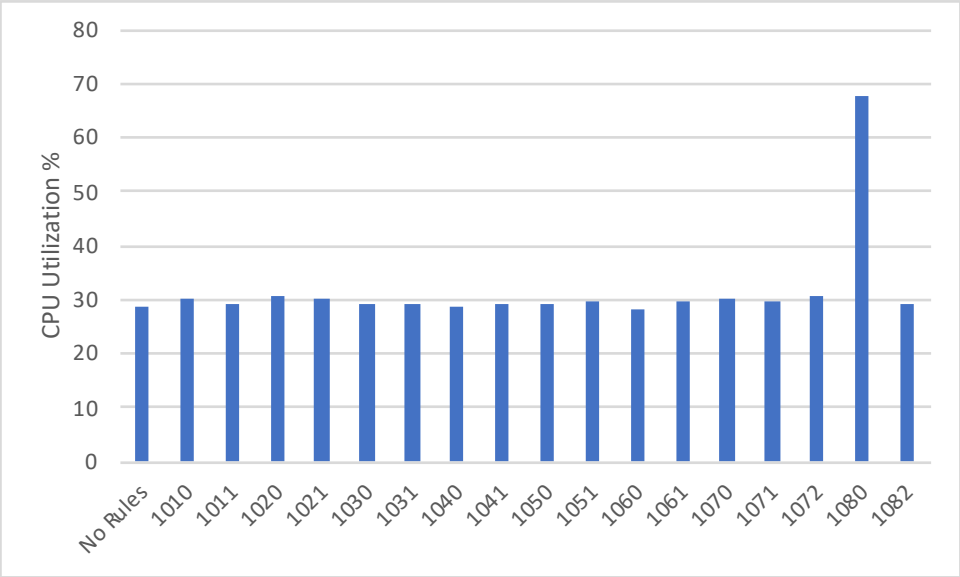

Figure 3 Average CPU utilization under each rule.

Figure 3 shows a graphical representation of the average CPU utilization during the 10-iteration test of each rule. Nearly every test with rules configured exhibited a higher CPU utilization than when there were no rules configured in Suricata. As discussed in Chapter 4, not every test was statistically significant enough to disprove the null hypothesis, but some were. Based on the statistically significant data points, detecting X.509 covert channels with

an intrusion detection system does have an effect on the performance of the system. The hypothesis has been supported with the gathered data.

TLS and TCP Rules

A number of rules were created that inspected the contents of the packet for specific identifiers, such as the start of a particular X.509 extension. Duplicate rules with the same content keyword were created, but the matching protocol was changed so one of the duplicate rules used TLS as the matching protocol, and the other used TCP as the matching protocol. This was done to study the possible performance impacts of TLS vs. TCP detection. It is important to note that X.509 certificates are transferred over the TLS application-layer protocol, and TLS is layered on top of the TCP transport-layer protocol (Dierks & Rescorla, 2008).

In total, six pairs of TLS and TCP rules were created and tested. Overall, with a 10-iteration test conducted for each rule, a total of 60 iterations were run for each TLS and TCP. When analyzing the rule performance details of all the TCP and TLS rules against each other, a very minor difference was found. TCP rules had a mean value of 4,458 ticks per check, and TLS rules had a mean value of 4,339 ticks per check. Figure 4 shows a graphical representation of the average number of ticks per check of each of the 12 total rules.

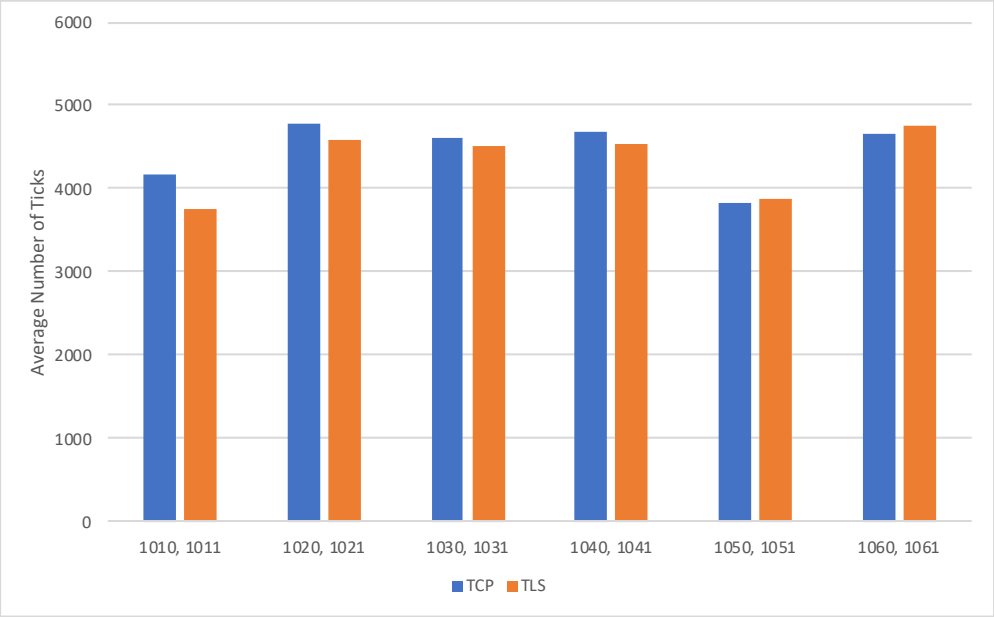

Figure 4 Comparing the average number of ticks between TLS and TCP rules.

As shown in the figure, two rules configured with the TLS protocol actually averaged a higher number of ticks per check over the 10 iteration tests. This suggests the mean data is not strong. To verify this assumption, a t-test was conducted comparing the TLS and TCP data. The t-value is 1.26161, and the *p*-value is 0.104789, short of the alpha value of 0.05. The comparison of TLS and TCP rulesets in this study is therefore not statistically significant.

Correct Length Detection

The X.509 covert channel proof-of-concept code developed by Reaves (2018) used the subject key identifier extension exclusively for carrying arbitrary data. As such, it was the initial focus of this study. Two additional extensions, the authority key identifier extension and the key usage extension were also studied. These three extensions all have common lengths which can be verified by Suricata. While the X.509 specification allows some flexibility in calculating the identifiers, a vast majority of certificates use a consistent method for this calculation which produces a constant length field. Nearly 99.8% of the over 1.2 billion certificates cataloged by Censys exhibited this feature (Durumeric et al., 2015).

In an effort to observe the performance impact of various rules, six different rules were created to verify these three extensions. These rules are broken down into two categories: rules that match the 3-byte extension identifier, and rules that match the 3-byte extension identifier and verify the 4-byte length description of the extension. The second set of rules will generate an alert if the extension exists, but the length of the field is incorrect. This is accomplished through searching for an exact match of the extension identifier and searching for an exact match of the length field. If the predefined length bytes are not found, an alert is generated.

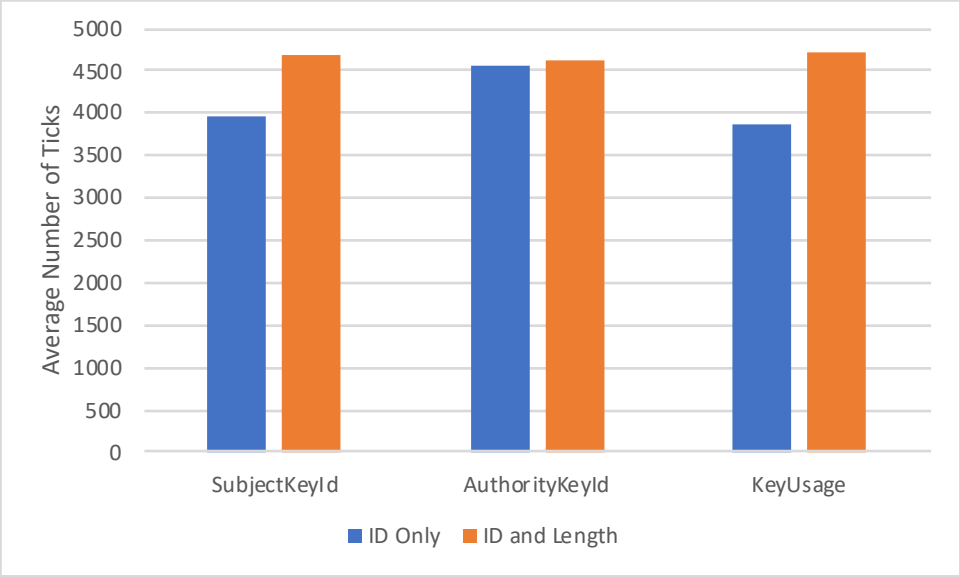

Figure 5 Comparing the average number of ticks between rules that only detect the identifier and rules that detect the identifier and verify the length of an extension.

Figure 5 shows a graphical representation of the average number of ticks per check that each of the six aforementioned rules consume. The three rules that only check for the 3-byte ID have a mean number of ticks of 4,128, and the three rules that check for the 3-byte ID and the lack of the predefined 4-byte length have a mean number of ticks of 4,670.

The rules that only check for the existence of the extension do not necessarily detect the existence of a covert channel, as regular legitimate certificates will also have the extension identifier if the optional extension is present (Cooper et al., 2008). Verifying the length is necessary to begin to verify the contents of the extensions. Simply alerting if the length is not correct is enough to accurately detect the X.509 covert channel created by Reaves (2018). Reaves' covert channel by default uses the subject key identifier field by placing as many as 10,000 bytes in the field, far more than the common length of 20. While the rules that detect a total of 7 bytes in the payload have a more significant CPU utilization than the rules detecting only 3 bytes, the additional data to detect in the payload is necessary for detecting X.509 covert channels.

As shown in the figure, all of the tests followed the same trend as described in the previous paragraph. A t-test was conducted on the data presented in this section to verify statistical significance. It was found that the data is in fact statistically significant. The t-value is 6.69173, and the *p*-value is < 0.00001, which is much less than the alpha value of 0.05, showing the data is in fact statistically significant.

Combined Rule

Each of the three sets of rules described in the previous section were then combined into a larger rule to further evaluate the impacts of larger rules. The three 3-byte rules to detect the presence of a specific extension's identifier were combined into a larger rule that searches packets for nine bytes, three bytes in each of three groups. Additionally, the rules that also detect incorrect extension lengths were combined into a larger rule that searches packets for a total of 21 bytes. The combined 3-byte rule has three separate content keywords which means Suricata must search the entire packet for the existence of each of the three

contents, in this case the identifier of the three extensions. The combined 3+4 byte rule searched for the same, in addition to also searching for the absence of the expected 4-byte length identifiers.

The mean number of ticks per check for rules that work with just one extension is 3,770, and the mean number of ticks per check for the combined rules that work with three extensions is 7,483. The combined rules have a more significant impact on the CPU and search for a greater number of bytes within the payload of the packet than the single rules. To verify the statistical significance, a t-test was computed between the combined rules and individual rules. The t-value is 14.01388, and the *p*-value is < 0.00001, which is less than the alpha value of 0.05. This data is statistically significant.

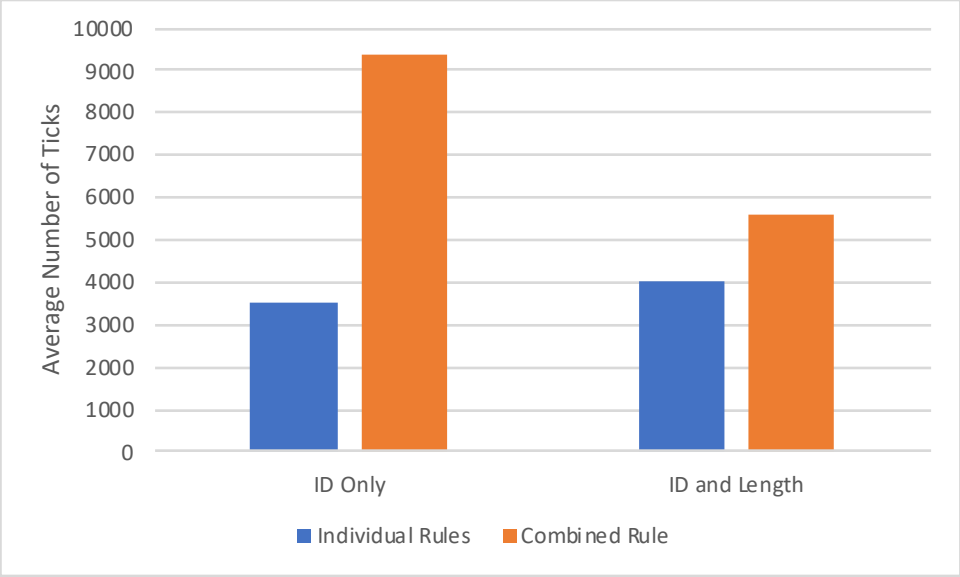

Figure 6 Comparing the average number of ticks between rules that detect one extension and combined rules that detect multiple extensions.

Figure 6 shows the difference in the average number of ticks consumed by the individual rules and the combined rules. The combined rules search for more content in the packets than the individual rules and they have a greater impact. Interestingly, the combined

rules that verify the length as well as the extension identifier have a smaller impact on the CPU than the combined rules that only match the extension identifier. Since these rules have sections that are negated – that is, the rule will match if a set of predetermined bytes are not found in the payload – Suricata may be able to short-circuit its searching if the first contents are not matched before searching for other content.

Lua Script

Occasionally the available Suricata rule options are not enough to fully verify the contents of a packet. For this reason, Suricata includes the ability to define custom lua scripts to perform additional detection functions (OSIF, 2016). A lua script was created to further verify the contents of the subject key identifier extension. The script first searches for the subject public key and the subject key identifier in the payload. After locating those two fields a SHA1 hash is taken of the subject public key. If that hash does not match the subject key identifier, an alert is generated.

This rule had a very significant impact on the CPU utilization of the system. The mean CPU utilization of the system with no rules configured was 28.9312%, compared to 67.7278% with the lua script rule configured. The average number of ticks per check of the lua rule was 8,122,695,314. This is a significant increase over all of the other rules tested. Of all of the other rules tested, the minimum average number of ticks was 3,754 and the maximum was 9,365.

The additional overhead demonstrated by this rule is quite significant, but fully verifying all of the contents of the subject key identifier extension requires additional features only afforded by a lua script. It is important to note that this script only verifies one extension

of the X.509 certificate. Thoroughly verifying as many components of the certificate as possible is likely to be extremely expensive if conducted through a custom lua script.

Recommendations

This study investigated the performance impacts of detecting X.509 covert channels in an intrusion detection system. A review of literature revealed the potential malicious impacts of covert channel usage as well as the need for detection of new malicious covert channels. As computer security is often thought of as a cat-and-mouse game between attackers and defenders, as attackers create new methods to carry out their goals, so to must the defenders work to detect these new methods.

After conducting this study, it has become clear that various IDS rules can have an impact on the performance of the system based upon how it is written. In some cases, this impact can be quite large depending upon what exactly is being detected. Through the literature review, the potential malicious impacts of covert channel usage are apparent.

Better Defined X.509 Extensions

In order to accurately detect X.509 extension misuse, there must be a clear definition of exactly what the extensions in X.509 certificates must look like, what data they must contain, and in what format. In the current specification, the contents of some extensions are not well defined. Additionally, the X.509 specification allows for the creation of custom extensions that do not already exist (Cooper et al., 2008).

In Reaves' X.509 covert channel proof-of-concept code, one specific extension was used to transfer arbitrary data. It turns out that this extension, the subject key identifier, can be calculated based upon another field in the certificate. This allows for real-time verification of

the contents of this extension making the detection of extension misuse possible. Other extensions are not as clearly defined or cannot be easily verified on the fly. When designing verification capabilities, the ability to quickly verify the data in a certificate extension needs to be a priority.

The feature that lead to accurate detection of the X.509 covert channel proof of concept code was the length of the extension. In the case of the subject key identifier extension, the most common length found was 20 bytes, while the covert channel placed much more data within the extension, upwards of tens of thousands of bytes. The problem, though, remains that the 20-byte length is not actually well defined in the X.509 specification. Therefore, this extension can still be technically correct and following the X.509 specification even if the length is not 20 bytes. The more well-defined each of the extensions can be in the specification, the easier it will be for defenders to detect extension misuse.

Efficient IDS Rules

IDS rule writers must continue to place the performance impacts of the rules they write as a high priority. The results of this study have shown that various rules created to detect the same item can have different impacts on the performance of the IDS. The literature review has shown that an overloaded IDS can lead to either smaller throughput of the system or the dropping of traffic, which can lead to not having the ability to detect true malicious attacks.

Future Research Recommendations

There is plenty of space in this field for additional future research. Future continuations of the themes in this study can take two major different paths: rule performance studies and covert channel detection studies.

Rule Performance

Even though much previous research has been conducted on the throughput and performance of various intrusion detection and prevention systems, there are still plenty of questions that can be answered with future research in this space. This study looked at intrusion detection system rule performance in the confines of detecting X.509 covert channels. Future studies can look at performance of rules beyond just this one simple use case.

A future study could critically examine the performance of existing rulesets that are in use in intrusion detection systems today. This study may attempt to pinpoint the specific rule options or features that have the highest cost in terms of resource utilization. The outcome of such research could inform the community as to which methods have a significant impact, allowing rule authors to more effectively create the most efficient rules possible.

The rule that had the most significant impact on the CPU utilization and therefore throughput of the system was the custom lua script. A lua script was necessary in this case to fully verify the subject key identifier extension due to content specific SHA1 hashing features being unavailable in existing Suricata rule options. Another avenue of future research takes place around the lua script. Again, in an effort to reduce overall rule performance impacts, researchers could survey existing rules that use custom lua scripts, paying close attention to the specific features that the script was relied upon for. If these features were known, Suricata

developers would be able to attempt to build them directly into the Suricata detection engine, hopefully in a more performant and efficient manner than a custom lua script can perform.

Covert Channel Detection

Attackers are often looking for methods of bypassing network security controls like firewalls and intrusion detection systems. Covert channels have the ability to transfer information hidden within legitimate protocols and can be one tool in the attacker or malware author's toolkit (Binsalleeh et al., 2014). This focused on just a few extensions in the X.509 certificate that could be used as a covert channel.

Future studies in covert channel detection should expand more fully upon the entire X.509 certificate to verify as many fields as possible. Previous research has theorized that additional fields and extensions could be utilized for a covert channel. Additionally, any protocol that is often allowed to pass through network firewalls can be a possible candidate for carrying a covert channel. While some research has been conducted on some of these protocols like DNS and ICMP, there is still room for additional research on detecting covert channels in these and new protocols.

Summary

This study brought forth literature that proved creating a covert channel using X.509 certificates is possible. This study also showed that this covert channel is detectable by verifying the extensions used in the channel to the greatest extent possible. Some extensions in the X.509 specification are too broad to write effective detection mechanisms for them.

The results of this study proved that rules written for an intrusion detection system can have an impact on the CPU utilization of the system. Literature has shown that this high

impact on the CPU can lead to a lower throughput of the system and the eventual dropping of network packets. This fact shows the importance of highly performant systems, especially as network bandwidth utilizations are on the rise.

Finally, this study showed that different methods of detecting packet features can have varied impacts on the performance of the system. For full verification of some network protocol features, additional utilities beyond that of the rule options available in the Suricata detection engine are necessary. In this case, a custom lua script was used, and was shown to have a significant impact on the CPU utilization of the system. As such, future research should be devoted to this problem of further verifying network traffic and detecting covert channels in a more performant manner.

References

Adom, D., & Joe, A.--agyem. (2018). Theoretical and conceptual framework : Mandatory ingredients Engineering, *8*(2), 8–16.

Afzal, Z., & Lindskog, S. (2015). Automated testing of IDS rules. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 1–2). IEEE. https://doi.org/10.1109/ICSTW.2015.7107461

Ahmad, W., & Qazi, A. S. (2018). Analysis of Interactive Utilization of CPU between Host and Guests in a Cloud Setup. *Computer Science and Engineering*, *8*(1), 7–15. https://doi.org/10.5923/j.computer.20180801.02

Albashear, A. M. M., Ali, H. A., & Ali, A. M. (2018). Detection of Man-in-the-Middle Attacks by Using the TCP Retransmission Timeout : Key Compromise Impersonation Attack as Study Case. In *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)* (pp. 1–8). IEEE. https://doi.org/10.1109/ICCCEEE.2018.8515845

Albin, E., & Rowe, N. C. (2012). A realistic experimental comparison of the Suricata and Snort intrusion-detection systems. *Proceedings - 26th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2012*, 122–127. https://doi.org/10.1109/WAINA.2012.29

Ansari, S., Hans, K., & Khatri, S. K. (2017a). A Naive Bayes classifier approach for detecting hypervisor attacks in virtual machines. In *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)* (Vol. 2018–Janua, pp. 1–6). IEEE. https://doi.org/10.1109/TEL-NET.2017.8343551

Ansari, S., Hans, K., & Khatri, S. K. (2017b). A Naive Bayes classifier approach for detecting

hypervisor attacks in virtual machines. In *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)* (Vol. 2018–Janua, pp. 1–6). IEEE. https://doi.org/10.1109/TEL-NET.2017.8343551

Arlos, P., & Fiedler, M. (2016). A Comparison of Measurement Accuracy for DAG, Tcpdump and Windump, (August).

Ayala, L. (2016). *Cybersecurity Lexicon*. *Cybersecurity Lexicon*. https://doi.org/10.1007/978-1-4842-2068-9

Bashir, U., & Chachoo, M. (2014). Intrusion detection and prevention system: Challenges & opportunities. In *2014 International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 806–809). IEEE. https://doi.org/10.1109/IndiaCom.2014.6828073

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network Anomaly Detection: Methods, Systems and Tools. *IEEE Communications Surveys & Tutorials*, *16*(1), 303–336. https://doi.org/10.1109/SURV.2013.052213.00046

Binsalleeh, H., Kara, A. M., Youssef, A., & Debbabi, M. (2014). Characterization of covert channels in DNS. *2014 6th International Conference on New Technologies, Mobility and Security - Proceedings of NTMS 2014 Conference and Workshops*. https://doi.org/10.1109/NTMS.2014.6814008

Brumen, B., & Legvart, J. (2016). Performance analysis of two open source intrusion detection systems. *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1387–1392. https://doi.org/10.1109/MIPRO.2016.7522356

Bulajoul, W., James, A., & Pannu, M. (2013). Network Intrusion Detection Systems in High-

Speed Traffic in Computer Networks. *2013 IEEE 10th International Conference on E-Business Engineering*, 168–175. https://doi.org/10.1109/ICEBE.2013.26

Callegati, F., Cerroni, W., & Ramilli, M. (2009). Man-in-the-middle attack to the HTTPS protocol. *IEEE Security and Privacy*, *7*(1), 78–81. https://doi.org/10.1109/MSP.2009.12

Carrara, B., & Adams, C. (2016). A Survey and Taxonomy Aimed at the Detection and Measurement of Covert Channels. *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security - IH&MMSec '16*, 115–126. https://doi.org/10.1145/2909827.2930800

Casalicchio, E., & Perciballi, V. (2017). Measuring Docker Performance. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion - ICPE '17 Companion* (pp. 11–16). New York, New York, USA: ACM Press. https://doi.org/10.1145/3053600.3053605

Chan, H., Hammad, E., & Kundur, D. (2016). Investigating the impact of intrusion detection system performance on communication latency and power system stability. In *Proceedings of the Workshop on Communications, Computation and Control for Resilient Smart Energy Systems - RSES '16* (pp. 1–6). https://doi.org/10.1145/2939940.2939946

Chen, J., Yao, S., Yuan, Q., He, K., Ji, S., & Du, R. (2018a). CertChain: Public and Efficient Certificate Audit Based on Blockchain for TLS Connections. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications* (pp. 2060–2068). IEEE. https://doi.org/10.1109/INFOCOM.2018.8486344

Chen, J., Yao, S., Yuan, Q., He, K., Ji, S., & Du, R. (2018b). CertChain: Public and Efficient Certificate Audit Based on Blockchain for TLS Connections. In *IEEE INFOCOM 2018 -*

IEEE Conference on Computer Communications (pp. 2060–2068). IEEE. https://doi.org/10.1109/INFOCOM.2018.8486344

Clark, J., & Van Oorschot, P. C. (2013). SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. *Proceedings - IEEE Symposium on Security and Privacy*, 511–525. https://doi.org/10.1109/SP.2013.41

Conti, M., Dragoni, N., & Lesyk, V. (2016). A Survey of Man in the Middle Attacks. *IEEE Communications Surveys and Tutorials*, *18*(3), 2027–2051. https://doi.org/10.1109/COMST.2016.2548426

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., & Polk, W. (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. https://doi.org/10.17487/rfc5280

Creswell, J. W., & Creswell, J. D. (2018). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (5th ed.). Sage.

Dacosta, I., Ahamad, M., & Traynor, P. (2012). Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 7459 LNCS, pp. 199–216). https://doi.org/10.1007/978-3-642-33167-1_12

Department of Defense. (1985). *Trusted computer system evaluation criteria*. *Department of Defense*. https://doi.org/DoD 5200.28-STD

Dierks, T., & Allen, C. (1999). *The TLS Protocol Version 1.0*. https://doi.org/10.17487/rfc2246

Dierks, T., & Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2, 1–104. https://doi.org/10.17487/rfc5246

Dietrich, C. J., Rossow, C., Freiling, F. C., Bos, H., Steen, M. Van, & Pohlmann, N. (2012). On botnets that use DNS for command and control. *Proceedings - 2011 7th European Conference on Computer Network Defense, EC2ND 2011*, 9–16. https://doi.org/10.1109/EC2ND.2011.16

Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., & Halderman, J. A. (2015). A Search Engine Backed by {I}nternet-Wide Scanning. In *22nd {ACM} Conference on Computer and Communications Security*.

Fu, P., Li, Z., Xiong, G., Cao, Z., & Kang, C. (2018). SSL/TLS Security Exploration Through X.509 Certificate's Life Cycle Measurement. In *2018 IEEE Symposium on Computers and Communications (ISCC)* (pp. 00652–00655). IEEE. https://doi.org/10.1109/ISCC.2018.8538533

Fuentes, F., & Kar, D. C. (2005). Ethereal vs. Tcpdump: A Comparative Study on Packet Sniffing Tools for Educational Purpose. *J. Comput. Sci. Coll.*, *20*(4), 169–176. Retrieved from http://www.ezproxy.dsu.edu:2108/citation.cfm?id=1047846.1047873

GReAT. (2015). A Fanny Equation: "I am your father, Stuxnet." Retrieved from https://securelist.com/a-fanny-equation-i-am-your-father-stuxnet/68787/

Harris, A., McGregor, J., Perencevich, E., Furuno, J., Zhu, J., Peterson, D., & Finkelstein, J. (2006). The Use and interpreatation of Quasi-Experimental Studies in Medical Informatics. *J Am Med Inform Association*, *13*(1), 16–23. https://doi.org/10.1197/jamia.M1749.Background

Heale, R., & Twycross, A. (2015). Validity and reliability in quantitative studies. *Evidence Based Nursing*, *18*(3), 66–67. https://doi.org/10.1136/eb-2015-102129

Hock, F., & Kortis, P. (2015). Commercial and open-source based Intrusion Detection System

and Intrusion Prevention System (IDS/IPS) design for an IP networks. In *2015 13th International Conference on Emerging eLearning Technologies and Applications (ICETA)* (pp. 1–4). IEEE. https://doi.org/10.1109/ICETA.2015.7558466

Homer, S., & Selman, A. L. (2014). {Turing} and the development of computational complexity. *{Turing}'s Legacy: Developments from {Turing}'s Ideas in Logic*, *42*, 299–328. https://doi.org/http://dx.doi.org/10.1017/CBO9781107338579.009

Hu, Q., Asghar, M. R., & Brownlee, N. (2017). Evaluating network intrusion detection systems for high-speed networks. *2017 27th International Telecommunication Networks and Applications Conference, ITNAC 2017*, *2017–Janua*, 1–6. https://doi.org/10.1109/ATNAC.2017.8215374

Jakobsen, S. K., & Orlandi, C. (2016). How To Bootstrap Anonymous Communication. *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science - ITCS '16*, 333–344. https://doi.org/10.1145/2840728.2840743

Kabir, M. F., & Hartmann, S. (2018). Cyber security challenges: An efficient intrusion detection system design. In *2018 International Young Engineers Forum (YEF-ECE)* (Vol. 4, pp. 19–24). IEEE. https://doi.org/10.1109/YEF-ECE.2018.8368933

Kaliski, B. S. (1993). A Layman's Guide to a Subset of ASN . 1, BER, and DER. Redwood City, CA: RSA Data Security, Inc.

Kaur, K., Singh, J., Kumar, K., & Ghumman, N. S. (2015). Programmable Firewall Using Software Defined Networking. *IEEE INDIACom*, 5–9.

Kemmerer, R. a. (1983). Shared resource matrix methodology: an approach to identifying storage and timing channels. *ACM Transactions on Computer Systems*, *1*(3), 256–277. https://doi.org/10.1145/357369.357374

Keppel, G., & Wickens, T. D. (2004). *Design and analysis: A researcher's handbook (4th ed.)*. *Technometrics*. https://doi.org/10.1198/tech.2005.s329

Khamphakdee, N., Benjamas, N., & Saiyod, S. (2014). Improving intrusion detection system based on Snort rules for network probe attack detection. *2014 2nd International Conference on Information and Communication Technology, ICoICT 2014*, 69–74. https://doi.org/10.1109/ICoICT.2014.6914042

Kuang, J., Mei, L., & Bian, J. (2012). An innovative implement in organizing complicated and massive intrusion detection rules of IDS. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems* (Vol. 3, pp. 1328–1332). IEEE. https://doi.org/10.1109/CCIS.2012.6664601

Kumar, R. (2014). *Research methodology : a step-by-step guide for beginners* (Fourth Edi). Sage. Retrieved from https://books.google.co.uk/books?id=MKGVAgAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Lampson, B. W. (1973). A note on the confinement problem. *Communications of the ACM*, *16*(10), 613–615. https://doi.org/10.1145/362375.362389

Leavy, P. (2017). *Research Design: Quantitative, Qualitative, Mixed Methods, Arts-Based, and Community-Based Participatory Research approaches*. *The sage Dictionary of social research methods*. https://doi.org/10.1177/0001699305050985

Levillain, O. (2012). One Year of SSL Internet Measurement, 11–20.

Leyden, J. (2011). Inside "Operation Black Tulip": DigiNotar hack analysed.

Leyden, J. (2012). Trustwave admits crafting SSL snooping certificate.

Loui, M. C. (1996). Computational complexity theory. *ACM Computing Surveys*, *28*(1), 47–

49. https://doi.org/10.1145/234313.234337

Mandiant. (2018). *M-Trends 2018*.

MD., N. C. S. C. F. G. G. M. (1993). *A Guide to Understanding Covert Channel Analysis of Trusted Systems*. Defense Technical Information Center. Retrieved from https://books.google.com/books?id=7JmaAQAACAAJ

Mohammed Nazer, G., & Lawrence Selvakumar, A. A. (2011a). Current Intrusion Detection Techniques in Information Technology - A Detailed Analysis. *European Journal of Scientific Research*, *65*(4), 611–624. Retrieved from http://www.europeanjournalofscientificresearch.com/ISSUES/EJSR_65_4_15.pdf

Mohammed Nazer, G., & Lawrence Selvakumar, A. A. (2011b). Current Intrusion Detection Techniques in Information Technology - A Detailed Analysis. *European Journal of Scientific Research*, *65*(4), 611–624.

Neupane, K., Haddad, R., & Chen, L. (2018). Next Generation Firewall for Network Security: A Survey. *Conference Proceedings - IEEE SOUTHEASTCON*, *2018–April*, 1–6. https://doi.org/10.1109/SECON.2018.8478973

Newman, R. C. (2007). Covert Computer and Network Communications Lecturer of Information Systems.

Onwuegbuzie, A. J. (2000a). Framework for Internal and External Validity. *Educational Research (AAER)*, *21*. Retrieved from http://files.eric.ed.gov/fulltext/ED448205.pdf

Onwuegbuzie, A. J. (2000b). Framework for Internal and External Validity. *Educational Research (AAER)*, *21*.

OSIF. (2016). Suricata User Guide.

Park, W., & Ahn, S. (2017). Performance Comparison and Detection Analysis in Snort and

Suricata Environment. *Wireless Personal Communications*, *94*(2), 241–252. https://doi.org/10.1007/s11277-016-3209-9

Pingle, B., Mairaj, A., & Javaid, A. Y. (2018). Real-world Man-in-the-middle (MITM) Attack Implementation Using Open Source Tools for Instructional Use. *2018 IEEE International Conference on Electro/Information Technology (EIT)*, 192–197. https://doi.org/10.1109/EIT.2018.8500082

Ponemon Institute. (2018). The 2018 Cost of a Data Breach Study by the Ponemon Institute. *IBM Security Services*, (July). Retrieved from https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=55017055USEN&

Postel, J. (1980). *User Datagram Protocol*. https://doi.org/10.17487/rfc0768

Postel, J. (1981). *Transmission Control Protocol*. https://doi.org/10.17487/rfc0793

Privitera, G. J. (2013). *Research Methods for the Behavioral Sciences*. SAGE Publications, Inc.

Reaves, J. (2018). Covert channel by abusing x509 extensions.

Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. https://doi.org/10.17487/RFC8446

Reynolds, J., & Postel, J. (1994). *Assigned Numbers*. https://doi.org/10.17487/rfc1700

Ritchey, P. C. (2015). Synthetic steganography : Methods for generating and detecting covert channels in generated media.

Roosa, S. B., & Schultze, S. (2013). Trust darknet: Control and compromise in the internet's certificate authority model. *IEEE Internet Computing*, *17*(3), 18–25. https://doi.org/10.1109/MIC.2013.27

Saboor, A., Akhlaq, M., & Aslam, B. (2013). Experimental evaluation of Snort against DDoS

attacks under different hardware configurations. *Conference Proceedings - 2013 2nd National Conference on Information Assurance, NCIA 2013*, 31–37. https://doi.org/10.1109/NCIA.2013.6725321

Salkind, N. (2010). Causal-Comparative Design. In *Encyclopedia of Research Design*. 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. https://doi.org/10.4135/9781412961288.n42

Satasiya, D., & Raviya Rupal, D. (2016). Analysis of Software Defined Network firewall (SDF). *Proceedings of the 2016 IEEE International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2016*, 228–231. https://doi.org/10.1109/WiSPNET.2016.7566125

Schaelicke, L., Slabach, T., Moore, B., & Freeland, C. (2003). Characterizing the performance of network intrusion detection sensors. *Recent Advances in Intrusion Detection (RAID)*, 155–172. https://doi.org/10.1007/978-3-540-45248-5_9

Scott, C. (2008). Network Covert Channels : Review of Current State and Analysis of Viability of the use of X . 509 Certificates for Covert Communications CURRENT STATE AND ANALYSIS OF VIABILITY OF THE USE OF X . 509. *Information Security*, (January).

Shah, S. A. R., & Issac, B. (2018). Performance comparison of intrusion detection systems and application of machine learning to Snort system. *Future Generation Computer Systems*, *80*(March), 157–170. https://doi.org/10.1016/j.future.2017.10.016

Soghoian, C., & Stamm, S. (2012). Certified lies: Detecting and defeating government interception attacks against SSL (short paper). In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in*

Bioinformatics) (Vol. 7035 LNCS, pp. 250–259). https://doi.org/10.1007/978-3-642-27576-0_20

Syta, E., Tamas, I., Visher, D., Wolinsky, D. I., Jovanovic, P., Gasser, L., … Ford, B. (2016). Keeping Authorities "honest or Bust" with Decentralized Witness Cosigning. *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, 526–545. https://doi.org/10.1109/SP.2016.38

Tesfatsion, S. K., Klein, C., & Tordsson, J. (2018). Virtualization Techniques Compared: Performance, Resource, and Power Usage Overheads in Clouds. *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, 145–156. https://doi.org/10.1016/j.otohns.2009.01.035

Titorenko, A. A., & Frolov, A. A. (2018). Analysis of modern intrusion detection system. In *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)* (Vol. 460–461, pp. 142–143). IEEE. https://doi.org/10.1109/EIConRus.2018.8317049

Uahhabi, Z. El, & Bakkali, H. El. (2017). An approach for evaluating trust in X.509 certificates. *2016 11th International Conference for Internet Technology and Secured Transactions, ICITST 2016*, 196–203. https://doi.org/10.1109/ICITST.2016.7856696

Vikan, D. E. (2015). *TLS and the future of authentication*. Norwegian University of Science and Technology.

Walsh, K. (2017). TLS with trustworthy certificate authorities. *2016 IEEE Conference on Communications and Network Security, CNS 2016*, (Spc), 516–524. https://doi.org/10.1109/CNS.2016.7860543

Warzynski, A., & Kolaczek, G. (2018). Intrusion detection systems vulnerability on

adversarial examples. *2018 IEEE (SMC) International Conference on Innovations in Intelligent Systems and Applications, INISTA 2018*. https://doi.org/10.1109/INISTA.2018.8466271

Wazan, A. S., Laborde, R., Chadwick, D. W., Barrere, F., & Benzekri, A. (2016). How Can I Trust an X.509 Certificate? An Analysis of the Existing Trust Approaches. *Proceedings - Conference on Local Computer Networks, LCN*, 531–534. https://doi.org/10.1109/LCN.2016.85

White, J. S., Fitzsimmons, T., & Matthews, J. N. (2013a). Quantitative analysis of intrusion detection systems: Snort and Suricata. *Cyber Sensing 2013*, *8757*, 875704. https://doi.org/10.1117/12.2015616

White, J. S., Fitzsimmons, T., & Matthews, J. N. (2013b). Quantitative analysis of intrusion detection systems: Snort and Suricata, (May 2014), 875704. https://doi.org/10.1117/12.2015616

Zander, S., Armitage, G., & Branch, P. (2007). A Survey of Covert Channels and Countermeassures in Computer Network protocols. *IEEE Communications Surveys*, *9*(3), 44–57. https://doi.org/10.1109/COMST.2007.4317620

Appendices

Appendix A: Suricata Rules

#1010 - SubjectKeyIdentifier ID in payload - tcp

alert tcp any any -> any any (msg:"Test Rule 1010 - SubjectKeyIdentifier ID"; content:"|55 1d 0e|"; sid:1010; rev:1; priority:1;)

#1011 - SubjectKeyIdentifier ID in payload - tls

alert tls any any -> any any (msg:"Test Rule 1011 - SubjectKeyIdentifier ID"; content:"|55 1d 0e|"; sid:1011; rev:1; priority:1;)

#----

#1020 - SubjectKeyIdentifier Exists with len NOT 20 bytes - tcp

alert tcp any any -> any any (msg:"Test Rule 1020 - SubjectKeyIdentifier ID Len"; content:"|55 1d 0e|"; content:!"|04 16 04 14|"; distance:0; sid:1020; rev:1; priority:1;)

#1021 - SubjectKeyIdentifier Exists with len NOT 20 bytes - tls

alert tls any any -> any any (msg:"Test Rule 1021 - SubjectKeyIdentifier ID Len"; content:"|55 1d 0e|"; content:!"|04 16 04 14|"; distance:0; sid:1021; rev:1; priority:1;)

#---

#1030 - AuthorityKeyIdentifer ID in payload - tcp

alert tcp any any -> any any (msg:"Test Rule 1030 - AuthorityKeyIdentifier ID"; content:"|55 1d 23|"; sid:1030; rev:1; priority:1;)

#1031 - AuthorityKeyIdentifer ID in payload - tls

alert tls any any -> any any (msg:"Test Rule 1031 - AuthorityKeyIdentifier ID"; content:"|55 1d 23|"; sid:1031; rev:1; priority:1;)

#---

#1040 - AuthorityKeyIdentifer Exists with len NOT 20 bytes - tcp

alert tcp any any -> any any (msg:"Test Rule 1040 - AuthorityKeyIdentifier ID Len"; content:"|55 1d 23|"; content:!"|04 18 30 16 |"; distance:0; sid:1040; rev:1; priority:1;)

#1041 - AuthorityKeyIdentifer Exists with len NOT 20 bytes - tls

alert tls any any -> any any (msg:"Test Rule 1041 - AuthorityKeyIdentifier ID Len"; content:"|55 1d 23|"; content:!"|04 18 30 16 |"; distance:0; sid:1041; rev:1; priority:1;)

#---

#1050 - KeyUsage ID in payload - tcp

alert tcp any any -> any any (msg:"Test Rule 1050 - KeyUsage ID"; content:"|55 1d 0f|"; sid:1050; rev:1; priority:1;)

#1051 - KeyUsage ID in payload - tls

alert tls any any -> any any (msg:"Test Rule 1051 - KeyUsage ID"; content:"|55 1d 0f|"; sid:1051; rev:1; priority:1;)

#---

#1060 - KeyUsage ID exists but wrong Bit String len in payload - tcp

alert tcp any any -> any any (msg:"Test Rule 1060 - KeyUsage ID"; content:"|55 1d 0f|"; content:!"|04 04 03 02|"; distance:0; sid:1060; rev:1; priority:1;)

#1061 - KeyUsage ID exists but wrong Bit String len in payload - tls

alert tls any any -> any any (msg:"Test Rule 1061 - KeyUsage ID"; content:"|55 1d 0f|"; content:!"|04 04 03 02|"; distance:0; sid:1061; rev:1; priority:1;)

#---

#1070 - Combined - Does not have ALL of SubjKeyID, AuthKeyID, KeyUsage
alert tls any any -> any any (msg:"Test Rule 1070 - BasicConstraints ID"; content:!"|55 1d 0e|"; content:!"|55 1d 23|"; content:!"|55 1d 0f|"; sid:1070; rev:1; priority:1;)

#1071 - Combined - Has all of SubjKeyID, AuthKeyID, KeyUsage, but not correct lengths.
alert tls any any -> any any (msg:"Test Rule 1071 - BasicConstraints ID"; content:"|55 1d 0e|"; content:!"|04 16 04 14|"; content:"|55 1d 23|"; content:!"|04 18 30 16 |"; content:"|55 1d 0f|"; content:!"|04 04 03 02|"; sid:1071; rev:1; priority:1;)

#1072 - Combined - Has all of SubjKeyID, AuthKeyID, KeyUsage, but not correct lengths, with distance 0 modifier
alert tls any any -> any any (msg:"Test Rule 1072 - BasicConstraints ID"; content:"|55 1d 0e|"; content:!"|04 16 04 14|"; distance:0; content:"|55 1d 23|"; content:!"|04 18 30 16 |"; distance:0; content:"|55 1d 0f|"; content:!"|04 04 03 02|"; distance:0; sid:1072; rev:1; priority:1;)

#--- LUA

#1080 - See if the Subject Key Identifier is the SHA1 of the subjectPublicKey
alert tls any any -> any any (msg:"Test Rule 1080 - SubjectKeyIdentifier ID"; lua:subjkey.lua; sid:1080; rev:1; priority:1;)

#1082 - If Subject Key Identifier is present, see if it is SHA1 of the subjectPublicKey
alert tls any any -> any any (msg:"Test Rule 1082 - SubjectKeyIdentifier ID"; content:"|55 1d 0e|"; lua:subjkey.lua; sid:1082; rev:1; priority:1;)

Appendix B: Custom Lua Script

```
function string.tohex(str)
   return (str:gsub('.', function (c)
      return string.format('%02X', string.byte(c))
   end))
end

function string.fromhex(str)
   return (str:gsub('..', function (cc)
      return string.char(tonumber(cc, 16))
   end))
end

function init (args)
   local needs = {}
   needs["payload"] = tostring(true)
   needs["packet"] = tostring(true)
   return needs
end

function match(args)
   a = tostring(args["payload"]):tohex()
   b = tostring(args["packet"]):tohex()
   local sha1 = require "sha1"


   --Find the subjectPublicKey
   if a:find("05000382010F00") then
      i, j = string.find(a, "05000382010F00")
      pubkeyinfo = string.sub(a, j+1, j+540)
```

```
            --55 1D 0E is identifier for subjkey, 04 16 04 14 is octet string -> octet string 20
bytes
            if a:find("551D0E04160414") then
               i, j = string.find(a, "551D0E04160414")
               subjkeyid = string.sub(a, j+1, j+40)
            end

            bytes = pubkeyinfo:fromhex()

            hashed = sha1.sha1(bytes)
            hashed = string.upper(hashed)

            if hashed == subjkeyid then
               io.write("\nMATCH SUCCESS\n")
               return 0
            end
            return 1
         end
         return 1
      end
      return 0
```

Appendix C: Packet Capture Files

The packet capture files used during all of the tests of this dissertation are located in an online github repository.

- wrccdc.pcap – This capture contains a collection of typical network traffic.
- malicious_text.pcap – This capture contains the transferring of a text file over the X.509 covert channel.
- mimikatz_exe.pcap – This capture contains the transferring of the mimikatz executable file over the X.509 covert channel.

The github repository is located at https://github.com/cmwelu/Dissertation