Automating Steps of Suricata Rule-Writing

by

Brian T. Carr

A Capstone Submitted to the Faculty of

Utica College

August 2021

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Cybersecurity

© Copyright 2021 by Brian T. Carr

All Rights Reserved

**Abstract**

Network Security Monitoring (NSM) is a practice used to defend a network of resources against threats. To detect new threats, signature-based IDS such as Suricata require rule development. Signature-based detection is the simplest method for detecting malicious activity. The verbosity of Suricata rules makes it hard for Suricata administrators to deploy new rules rapidly. The OISF (2021) acknowledges that most Suricata administrators are only using the default ruleset. Signature-based security solutions such as Suricata rely solely on their rulesets for their detection capabilities. The signatures contained within these rulesets define the anomalies which generate alerts. This research project provided a Python script that can be used from a command-line interface to output syntactically valid Suricata rules. The author intended that Suricata administrators and others would use the Python script developed during this research project to assist the rapid deployment of novel Suricata rules.

*Keywords: Network Security Monitoring, Intrusion Detection Systems, Signatures, Signature-Based Analysis, Python, Suricata, Open Information Security Foundation*

Table of Contents

[List of Illustrative Materials iv](#_Toc78910418)

[Automating Steps of Suricata Rule-Writing 1](#_Toc78910419)

[Purpose Statement 2](#_Toc78910420)

[Research Questions 3](#_Toc78910421)

[What is Network Security Monitoring? 3](#_Toc78910422)

[What is a Signature? 4](#_Toc78910423)

[Intrusion Detection Systems 5](#_Toc78910424)

[Signature-Based Network Security Monitoring Limitations 5](#_Toc78910425)

[Encrypted Network Traffic 6](#_Toc78910426)

[Value of Rulesets in Network Security Monitoring Solutions 7](#_Toc78910427)

[Scripting and Automation Potential 8](#_Toc78910428)

[Literature Review i](#_Toc78910429)

[Network Security Monitoring Background and Definition 8](#_Toc78910430)

[The History of Network Security Monitoring 9](#_Toc78910431)

[Purpose and Value of Network Security Monitoring 10](#_Toc78910432)

[Intrusion Detection and Intrusion Prevention Solutions 11](#_Toc78910433)

[Intrusion Detection Systems 11](#_Toc78910434)

[Intrusion Prevention Systems 11](#_Toc78910435)

[Network-Based vs. Host-Based Solutions 12](#_Toc78910436)

[Intrusion Detection Systems 12](#_Toc78910437)

[Signature-Based Intrusion Detection Systems 13](#_Toc78910438)

[Anomaly-based Intrusion Detection Systems 13](#_Toc78910439)

[Suricata 14](#_Toc78910440)

[Suricata Rules 14](#_Toc78910441)

[Open Informaiton Secuity Foundation 15](#_Toc78910442)

[Signatures and Signature-Based Security Solutions 15](#_Toc78910443)

[Network Traffic 16](#_Toc78910444)

[Data Encapsulation 19](#_Toc78910445)

[Developing Signatures for Encrypted Traffic 20](#_Toc78910446)

[The Value of Rulesets in Signature-Based Network Security Monitoring Solutions 21](#_Toc78910447)

[Default Rulesets 21](#_Toc78910448)

[The Shortcomings of Signature-Based Network Security Monitoring 22](#_Toc78910449)

[False-Negative Events: Missed Alerts 22](#_Toc78910450)

[The Importance of Rule-Writing 23](#_Toc78910451)

[Discussion of Findings 24](#_Toc78910452)

[The Value of Network Security Monitoring 24](#_Toc78910453)

[The Value Rulesets of Signature-Based IDS 24](#_Toc78910454)

[Signature-Based IDS Limitations 25](#_Toc78910455)

[Simplifying the Rule Writing Process 25](#_Toc78910456)

[Generating Suricata Signatures with Suri-rule-gen.py 26](#_Toc78910457)

[Future Research and Recommendations 27](#_Toc78910458)

[Conclusion 27](#_Toc78910459)

[References 29](#_Toc78910460)

# List of Illustrative Materials

Figure 1 – Suricata Rule Syntax 4

Figure 2 – IPv4 Packet Structure 16

Figure 3 – Suricata Packet Processing using autofp Runmode and Multiple Capture Threads 18

Figure 4 – OSI Model Showing Data Transmission 19

Figure 5 – Packet Encapsulation Visualization 20

# Automating Steps of Suricata Rule-Writing

"The current state of security for Internet-connected systems makes me think of the Wild West (Sanders & Smith, 2014)." The Internet Crime Complaint Center reported that from 2016-2020 victims in the United States experienced a combined total of $13.3 billion in total losses from cybercrime (IC3, 2021). All organizations that are networked to the internet are likely to experience a breach (Bejtlich, 2013). Many security solutions defend against cybercrime and other malicious activity. At the network level, Intrusion Detection Systems (IDS) can identify and alert on desired activity. Signature-based detection is the simplest method of detecting malicious activity. Signature-based detection solutions focus on matching known malicious indicators to newly supplied data (Scarfone & Mell, 2007). One popular signature-based network security tool is Suricata. Suricata is a network-based IDS, Intrusion Prevention System (IPS), and Network Security Monitoring (NSM) solution that protects many different networks (OISF, 2021).

This research project aimed to develop a tool that automates steps in the Suricata rule-writing process. The author's intention for the tool developed in this research project was to assist Suricata administrators in rapidly deploying Suricata rules. This project contains information obtained from scholarly articles, textbooks, and official Suricata documentation and training resources. The tool developed was intended to be used by Security Analysts, Network Administrators, Detection Engineers, Malware Analysts, Incident Response Analysts, and all other Suricata administrators. Additionally, the information comprising this project is relevant to all NSM administrators.

Computers have become embedded in all facets of every industry. From hospitals to federal government agencies, every organization has found a reason to utilize information technology systems in their daily activities. These computers and the networks they comprise are valuable. Although, they are not arbitrarily secure. While firewalls attempt to block unwanted connections, they have limited alerting mechanisms if something malicious has occurred. NSM solutions address this problem by generating alerts based upon the rulesets they utilize. Alhomoud et al. (2011) explained that NSM technologies such as network-based IDS systems are necessary for detecting malicious activity at the network level. The 13th control from the Center for Internet Security (CIS) CIS Controls V8 (2021) is the ability to operate processes and tooling to practice network monitoring for security threats across an organization's network. Organizations can implement this control through the organizational practice of NSM.

## Purpose Statement

The purpose of this research project was to develop tools that automate steps in the Suricata rule-writing process. The general problem is that signature-based NSM solutions such as Suricata require administrators to write rules to identify new anomalies (Einay et al., 2021). The specific problem is that many Suricata administrators and analysts dealing with NSM solutions find that the rules utilized by the Suricata open-source NSM solution are challenging to develop. The Open Information Security Foundation (OISF) (2021) acknowledged that most Suricata administrators are only using the default rulesets. It is possible that if Suricata rules became easier to write, more Suricata administrators would develop custom rules.

Additionally, signature-based NSM solutions heavily rely on their rulesets for functionality (Einay et al., 2021). Sanders (2014) stated that one issue plaguing NSM is that it requires multiple personal with specialized expertise to implement effectively. While less experienced junior analysts can practice NSM, guidance from a senior analyst is required (Sanders, 2014). Additionally, rule-developers currently require a wide range of skills to write valuable signatures which cover attack variations while still being efficiently utilized by the IDS engine (Shipulin, 2018). The CIS Controls V8 also stated that through security operation, organizations would locate tactics, techniques, and procedures of attackers in addition to IOCs that are useful in the detection process of network monitoring (CIS, 2021). The tool developed during this research project was designed to be used in the rapid deployment of Suricata signatures.

## Research Questions

The research contained within this project attempted to answer the questions: 1) Why is NSM valuable? 2) How necessary are signatures in signature-based NSM solutions? and 3) How can rule-writing for Suricata be simplified for Suricata administrators?

## What is Network Security Monitoring?

Bejtlich (2013) described NSM as a threat-centric approach to detecting and responding to specified anomalies. NSM methodology and NSM solutions are valuable in any network environment. In most NSM solutions, an IDS generates alerts when it detects an anomaly. NSM centers on the idea that prevention eventually will fail; subsequently, NSM involves detection methods and not prevention methods (Sanders & Smith, 2014). Network-based NSM solutions lie between the network segment which it is monitoring and the network egress point. The NSM solution receives network traffic via a port mirroring solution (sometimes referred to as SPAN ports) or a network tap (hardware or software device used to duplicate network traffic). The IDS engine can compare the traffic to the ruleset rules and match the desired anomalies by obtaining this traffic. While NSM is a methodology that goes beyond solutions such as Suricata, the NSM solution handles the core functionality of detecting and alerting upon specified network events (Bejtlich, 2013).

Signature-based NSM solutions allow for entities to protect against the threats identified within their rulesets. These rulesets contain signatures for cyberthreats such as ransomware or trojan malware infections, phishing sites, scanning activity, application use, and much more. NSM solutions are valuable because they can detect threats related to the commission of cybercrime or other nefarious activities.

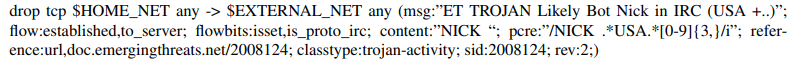
### What is a Signature?

Joshi et al. (2012) explained that a signature is a piece of data that can distinguish data. Signatures contain a combination of strings of characters, hex values, or various other static values. Joshi et al. stated that intrusion detection systems utilize signatures to locate security concerns in network traffic (2012).

Suricata uses rules to identify signatures. Rule files contain rules used by the Suricata engine, and the rule files have a .rules extension. These rule files contain the signatures designed for the Suricata engine to utilize when processing network traffic. Figure 1shows a sample Suricata rule. The sample Suricata rule contains an action, a header that defines the protocol, addresses, and ports involved in the rule, and the rule options that make the rule more specific (OISF, 2021).

**Figure 1**

*Suricata Rule Syntax*



*Note.* Image obtained from official Suricata documentation (OISF, 2021).

## Intrusion Detection Systems

Andress (2019) described IDS as software or hardware solutions that monitor networks, hosts, or applications for unauthorized activity. The Suricata engine monitors network traffic looking for specified events. The Suricata engine can be tuned to ignore certain events or even detect innocuous events by tuning the ruleset which it utilizes (OISF, 2021). Broucek and Turner (2004) described the two types of IDS which can monitor traffic at a network level. The two types of IDS are anomaly-based solutions and signature-based solutions. Anomaly-based IDS solutions function by detecting unusual or abnormal behavior by comparing current events to a baseline of the system's standard behavior. This method typically has a higher rate of false positives. On the other hand, signature-based methods compare traffic to a database of rules which contain signatures of previously identified malicious indicators (Broucek & Turner, 2004).

Sikorski and Honig (2012) explained that IDS solutions and other security appliances allow network administrators to employ signature-based defenses against malicious content. Commonly located Indicators of Compromise (IOC) include Uniform Resource Locators (URL) and Internet Protocol (IP) addresses. These IOCs frequently appear in network traffic and subsequently make for good NSM signatures. However, these signatures are often not valuable for very long because malicious actors regularly migrate infrastructure to evade security solutions (Sikorski & Honig, 2012). The short lifespans of these campaign-specific IOCs emphasize the need for organizations to develop and implement signatures for novel threats in a minimal amount of time.

## Signature-Based Network Security Monitoring Limitations

Signature-based NSM solutions have a significant shortcoming; They cannot detect anomalies which they do not have signatures for. Signature-based NSM solutions are only as valuable as the signatures which they are utilizing in their rulesets. Additionally, while reputable and mature organizations develop default rulesets for IDS systems such as Suricata, they may be missing novel threats. Minimizing the time between IOC identification and signature deployment shrinks the risk associated with the threat that the signature identifies (Andress, 2019). The Open Information Security Foundation (2021) reported that most Suricata administrators only utilize preexisting rulesets. These entities could directly benefit from writing and implementing signatures for unaccounted threats that do not exist within the default rulesets. Entities that only enable the default rulesets will only detect anomalies where other researchers have already developed signatures (OISF, 2021).

### Encrypted Network Traffic

Transport Layer Security (TLS) and Secure Sockets Layer (SSL) are protocols used to encrypt communications for everyday applications. Both legitimate applications and malware utilize encryptions protocols to encrypt their communications. To combat the threats associated with encrypted network traffic, researchers at Salesforce developed solutions that identify encrypted TLS communications by providing them with a unique value that can be used as a fingerprint. These solutions were JA3 and JA3S. These solutions work by using the MD5 message-digest algorithm on a uniquely formatted string that contained values from the TLS Client Hello and TLS Server Hello packets to generate a hash value that can identify future TLS communications (Althouse, 2019).

Papadogiannaki and Ioannidis (2021) report that over 75% of worldwide internet traffic is encrypted. When an encryption protocol such as SSL/TLS encrypts TCP level traffic, it does not encrypt the IP packet header. This encrypted traffic poses a problem for NSM solutions as many signatures identify malicious content within the packets of network traffic. If the traffic under analysis by the IDS is encrypted, the content that these rules would match on is obfuscated, minimizing the effectiveness of content-based signatures in the face of encrypted traffic. Papadogiannaki and Ioannidis addressed this in their IDS solution known as HeaderHunter. HeaderHunter efficiently analyzes encrypted network traffic by focusing only on analyzing the packet header and metadata. This significantly limits potential signature development, signatures that look for information within the packet header or metadata will have value when applied to encrypted traffic. Additionally, HeaderHunter functions in a way that made signature generation easy for administrators (Papadogiannaki & Ioannidis, 2021).

## Value of Rulesets in Network Security Monitoring Solutions

Signature-based IDS engines such as Suricata function by comparing packets of network traffic flowing across a network to rules contained within rule files. Andress (2019) explained that signature-based IDS solutions have similar functionality to signature-based antivirus solutions. Both types of security applications store signatures in a file or database, which allows for the solutions to detect the desired anomalies. Additionally, both signature-based NSM and antivirus have the same shortfall of only detecting anomalies with existing signatures. Novel threats may go unidentified, severely limiting the effectiveness of signature-based solutions (Andress, 2019). To combat this shortfall, this research project's author suggests that rule-writing has become a necessary skill of any IDS administrator.

Ganesan et al. (2019) described that there are *located* threats and *novel* threats. These novel threats have not yet been located currently in existence. Additionally, Ganesan et al. explained that if a threat changes in the slightest manner, it may require the signature to be adjusted to catch this change. The constant need for signature updating and management is one of the critical factors that lead to this project's development. Failure to quickly adjust rulesets can result in false negatives (situations where an alert should have been generated but was not).

## Scripting and Automation Potential

The scripts contained within this research aim to automate the generation of signatures that matched data in the packet header or packet metadata. In a training video produced by the OISF and Applied Network Defense (2017), the author explained that signatures designed to match DNS calls and other regularly occurring events are straightforward to develop and have the potential to be scripted. A focus on IP packet headers and metadata was applied when developing the code for this research project.

# Literature Review

## Network Security Monitoring Background and Definition

In the book *Practical Network Security Monitoring,* Bejtlich (2013) explained that the principles of NSM are collection, analysis, and incident escalation upon detection. Bejtlich additionally explained that NSM is a methodology that can allow entities to detect intrusions and minimize their opportunity to wreak havoc (2013). Andress (2019) explained that an entity could detect malicious activity by auditing the information in an environment. Andress explained that IDS solutions are monitoring and alerting tools that can generate alerts (2019).

In the book, *Applied Network Security Monitoring,* Sanders and Smith (2014)explained that the practice of network security monitoring has three main functions. Those functions were collection, detection, and analysis. Sanders and Smith explained that collection could involve various data types, from full packet captures to simply capturing the alert data. The type and amount of data being the NSM is collecting will often change on an organizational basis. Sanders and Smith explained that the detection function of NSM is the function related to the signatures within the rulesets that the IDS system is utilizing. Sanders and Smith described the analysis stage of NSM as having an analyst investigate and interpret the data located during the detection stage. Various types of analysis can occur at the analysis stage of NSM. Some types of analysis include packet analysis, network forensics analysis, host forensic analysis, and malware analysis. The analysis is the most time-consuming of the three stages as it involves engaging incident response personal (Sanders & Smith, 2014).

In a research article written by Einay et al. (2021), the authors explained that one of the primary weaknesses of many security solutions is their inability to detect intrusions at a network level. Einay et al. went on to explain that IDS can be used to detect intrusions at a network level, subsequently accounting for this weakness. IDS systems utilize inline network traffic to detect many types of anomalies (Einay et al., 2021).

In a post on their website, the Center for Internet Security (2018) explained that they offer a NSM solution known as Albert. CIS further explained that Albert is a passive IDS that detects malicious activity by generating alerts from threat signatures. The Albert IDS system utilizes the Suricata engine to facilitate this detection capability (CIS, 2018). In a blog on their website, CIS explains that State, Local, Tribal, and Territorial government entities utilize Albert to detect malicious activity in a timely manner. CIS additionally stated that NSM plays an essential role in an organization's defense-in-depth strategy (CIS, 2021b). The CIS Controls V8 13th control, Network Monitoring and Defense, highlights the need to monitor for threats at a network level (CIS, 2021a).

### The History of Network Security Monitoring

In Richard Bejtlich's (2013) book The Practice of Network Security Monitoring, Bejtlich provided the history of Network Security Monitoring. Bejtlich stated that NSM informally began when Todd Heberlein developed an application named the Network Security Monitor in 1988. Heberlein's Network Security Monitor was the first IDS system to utilize data traffic going across the network as the data being used to generate alerts. According to Bejtlich, in 1993, Heberlein worked with the Air Force Computer Emergency Response Team (AFCERT) to deploy another version of the Network Security Monitoring named the Automated Security Incident Measurement (AISM) system. Bejtlich explained that he codified NSM's definition during a webcast for SearchSecurity in 2002 (2013).

Sanders and Smith (2014) explained that prior to the conceptualization of NSM, the methodology of locating malicious threats at a network level was known as intrusion detection. Sanders and Smith further stated that intrusion detection is a vulnerability-centric approach, while NSM is a threat-centric approach. The goal of NSM is to detect in hopes of responding more time efficiently (Sanders & Smith, 2014).

### Purpose and Value of Network Security Monitoring

In the book *Practical Malware Analysis,* Sikorski and Honig (2012) described how signature-based IDS and IPS solutions allow network administrators to employ content-based countermeasures. These solutions often allow analysts to write signatures for IP addresses or URLs, making them ideal for network-wide detection. By writing signatures to match upon desired traffic anomalies, security and incident response personnel are now able to protect against previously identified threats (Sikorski & Honig, 2012).

In an article published within the Procedia Journal of Computer Science, Alhomoud et al. (2011) explained that IDS solutions are not an essential component of any entity's environment. IDS solutions have the functionality of detecting malicious traffic going across the network. Alhomoud et al. added that organizations are currently in need of valuable detection systems (2011).

## Intrusion Detection and Intrusion Prevention Solutions

Sanders (2017) explained that NSM does not involve prevention as preventative measures eventually fail. Instead, the NSM methodology focuses on the detection of anomalous activity. While IPS systems can be used by teams implementing the NSM practice, NSM deals exclusively with IDS systems (Sanders, 2017). Andress (2019) explained that IDS could only monitor and detect. IDS cannot take any direct action preventing or altering the flow of network traffic. On the other hand, IPS systems sit in line with the network traffic (Firewalls also must sit inline), allowing them to take actions depending on what the engine is utilizing for a ruleset (Andress, 2019).

### Intrusion Detection Systems

Andress (2019) described IDS solutions as hardware or software tools that can monitor hosts, networks, and applications for desired anomalies. Andress then explained that there are two types of IDS solutions, signature-based and anomaly-based. Sanders and Smith (2014) explained that signature-based IDS are the most common. Sanders and Smith stated that signature-based IDS function by analyzing packets of network traffic for signatures derived from IOCs. In signature-based IDS, when a packet of network traffic matches a signature, an alert is generated. Sanders and Smith additionally report that modern IDS solutions can detect malicious activity that goes beyond the scope of traditional malware infections (Sanders & Smith, 2014).

### Intrusion Prevention Systems

Andress (2019) describes IPS solutions as security solutions that often take data sent by the IDS to take action on desired anomalies. An example of an IPS taking action would be rejecting traffic from a specified IP address. The IDS solution, on the other hand, could only locate the traffic and generate an alert. IDS cannot take action as they sit out of line with the network egress point (Andress, 2019).

## Network-Based vs. Host-Based Solutions

Broucek and Turner (2004) stated that there are four types of intrusion detection systems, host-based IDS (HIDS), application-based IDS (AIDS), stack-based IDS, and network-based IDS (NIDS). Vacca (2014) indicated that network-based IDSs monitor the packets of network traffic traveling across a network for signs of malicious or desired activity. Vacca additionally states that the IDS system can potentially identify attacks before reaching their targets (Vacca, 2014).

Sikorski and Honig (2012) described how IOCs such as IP addresses and URLs are effective in defending against malware and malicious activity. However, these signatures may only be of value for a short time. The limited lifespan of these signatures is due to the malicious actor's ability to quickly change infrastructure to evade security solutions (Sikorski & Honig, 2012).

## Intrusion Detection Systems

In an article published in *Network Security,* Shipulin (2018) explained that many organizations utilize IDS solutions to control and inspect the traffic inside of the network. These solutions allow organizations to investigate what type of traffic flows across their network (Shipulin, 2018). Broucek and Turner (2004) explained that both signature and anomaly-based IDS systems detect anomalies and malicious activity. Signature-based IDS constantly compare the collected data is against a database of known attacks and vulnerabilities. Regarding anomaly-based IDS, the system attempts to identify abnormalities or changes in behavior instead of the system's everyday activities. Anomaly-based IDS are subject to a more significant number of false-positive alerts than signature-based IDS solutions (Broucek & Turner, 2004).

In *Applied Network Security Monitoring,* Sanders and Smith (2014) explained that an anomaly is an event in a system or network that is observable and concerned unusual. Detection tools such as IDS generate alerts when an anomaly is located.

### Signature-Based Intrusion Detection Systems

In the book *Foundations of Information Security*, Andress (2019) explained that signature-based IDS systems function similarly to signature-based antivirus (AV) solutions. Both signature-based IDS and AV utilize a database of signatures and compare data to those databases. Regarding AV solution, this data might be the file on a hard drive or an outbound web connection. Regarding network-based IDS, this data is network traffic flowing to and from the network egress point (Andress, 2019).

Joshi et al. (2012) explained that a signature is a portion of data used to distinguish that specific data. Signatures contain a combination of a string of characters, hex values, or various other static values. Joshi et al. further explained that intrusion detection systems utilize signatures to locate security concerns in network traffic (Joshi et al., 2012). Shipulin (2018) stated that signature sets are pivotal in detecting known attack vectors.

### Anomaly-based Intrusion Detection Systems

Andress (2019) explained that anomaly-based IDS work by developing a normal network profile and alerting when the network deviated from the profile. Some of the data collected by the anomaly-based IDS include the amounts and type of traffic being sent over the network. Andress additionally acknowledged that anomaly-based IDS tend to produce more false-positive alerts than signature-based IDS. Broucek and Turner (2004) stated that anomaly-based intrusion detection systems generally have a higher rate of false-positive alerts generated when compared to signature-based IDS.

## Suricata

Alhomoud et al. (2011) stated that Suricata is a signature-based IDS/IPS solution that utilizes rulesets to alert or act upon rule matches in sniffed traffic. Suricata is a networked-based solution; other types of IDS solutions include host-based or application-based IDS solutions. The Open Information Security Foundation (OISF) developed Suricata with funding from the Department of Homeland Security's Directorate for Science and Technology Homeland Security open technology (HOST) program. The OISF released Suricata in July of 2010 (Alhomoud et al., 2011).

In the official Suricata documentation provided by the OISF, they defined Suricata as a well-functioning network IDS, IPS, and NSM engine. Suricata utilizes threads, thread-modules, and queues to process the network packets that are provided to the engine. Suricata additionally has three types of thread-modules. These thread-module types are decoding, detecting, and output modules. Each thread is able to utilize one or more thread-modules. The ways that the Suricata engine can be configured to utilize threads, thread-modules, and queues are known as runmodes.

### Suricata Rules

In the official Suricata documentation provided by the OISF (2021), rulesets and the signatures they contain are critical to the functionality of the Suricata engine. The rules utilized by the Suricata engine have three main sections. The first section is the rule action that determines what action the Suricata engine takes when the alert becomes triggered by a match. The second section is the header which defines the protocol, IP addresses, ports, and directionality of the rule.

The OISF (2021) discussed how systems running Suricata could be configured to capture network traffic. The official Suricata documentation stated that Suricata could be configured to capture packets in a variety of modes. These modes include pcap, pfring, and afpacket. Each of these modes captures network traffic slightly differently (OISF, 2021).

### Open Information Security Foundation

Suricata is developed and maintained by the OISF, are a non-profit organization. The OISF licensed the Suricata source code under version 2 of the GNU General Public License (GPL). Suricata is open source, and the source code is available on GitHub. Suricata can be installed on various operating systems, including Linux-based OS such as Ubuntu or Red Hat Enterprise Linux, Mac OS X, and Windows (OISF, 2021).

## Signatures and Signature-Based Security Solutions

Andress (2019) stated that signature-based IDS function similarly to signature-based antivirus programs. Both types of solutions utilize a database that contains signatures that match desired anomalies. Shipulin (2018) explained that IDS solutions, AV solutions, and Web Application Firewall (WAF) solutions utilize signature-based detection methods.

In Special Publication 800-94 published by the Nation Institute for Standards and Technology (NIST), Scarfone and Mell (2007), explained that signature-based detection methods are the simplest detection methods to implement as they compare data from a packet, log entry, or another source to predefined values in order to locate matches. Scarfone and Mell stated that false-positive events are more common in anomaly-based detect solutions as profile training is inherently tricky (2007).

## Network Traffic

In *Applied Network Security Monitoring,* Sanders and Smith (2014) explained that devices have the ability to communicate once networked together as long as they utilize a common networking protocol. Sanders and Smith further explained that the TCP/IP protocol stack is what facilitates Internet-connected devices. The TCP/IP data is transmitted within an Ethernet packet using a process called data encapsulation. The Ethernet data is transmitted with a physical layer protocol and contains data for higher layer protocols such as TCP/IP, DNS, HTTPS, and more (Sanders & Smith, 2013). Figure 2 shows the structure of the data within an IPv4 packet.

**Figure 2**

*IPv4 Packet Structure*

Graphical user interface

Description automatically generated with medium confidence

*Note.* Image obtained from *Applied Network Security Monitoring* (Sanders & Smith, 2013).

Sanders and Smith (2014) explained that the Suricata engine takes network packets from a monitoring port, decodes them, analyzes them utilizing the ruleset, and generates any alerts located in the packets under analysis. Sanders and Smith explained that the Suricata IDS engine goes through multiple steps before finally generating an alert. First, the packet is obtained off of the monitoring port and acquired. Next, Suricata attempts to decode information from the network packets. Then Suricata utilizes the detection engine to detect any signature matches before finally providing output (Sanders & Smith, 2014).

The official Suricata documentation elaborates on the various packet processing procedures for each of the available Suricata runmodes. Most runmodes utilize both packet capture threads and packet processing threads. Runmodes such as workers or single do not utilize packet capture threads but instead pass the packet directly to the packet processing threads (OISF, 2021). Figure 3 shows the Suricata packet processing method when configured to use the autofp runmode with multiple capture threads.

**Figure 3**

*Suricata Packet Processing using autofp Runmode and Multiple Capture Threads*

Diagram

Description automatically generated

*Note.* Image obtained from official Suricata documentation (OISF, 2021)

In the book *Practical Packet Analysis,* *Sanders* (2017) described how network data exists at multiple layers of what is known as the OSI model. The OSI model is a seven-layer model that explains what state the data is in during different transmission phases. The OSI model goes from the first layer, which is physical, to the application layer, which utilizes application-specific protocols (Sanders, 2017). Figure 4shows a graphic showing how data traverses the OSI model before being transmitted.

**Figure 4**

*OSI Model Showing Data Transmission*

Diagram

Description automatically generated

*Note.* Image obtained from *Practical Packet Analysis* (Sanders, 2017).

### Data Encapsulation

Sanders (2017) explained that data encapsulation allows for data to go through the OSI model. Each layer of the OSI model adds a header or footer, allowing the system to know how to communicate. Sanders stated that the entire packet with all protocol headers is passed across the network medium at layer one of the OSI model. In the process of data encapsulation, each layer of the OSI model is only able to communicate with the protocol above and below them. The layer one traffic, which contains all the header and footer values for each additional layer, is formally known as a protocol data unit (PDU). Figure 5shows a graphic displaying what data encapsulation looks like at each layer of the OSI model.

**Figure 5**

*Data Encapsulation Visualization*

Diagram

Description automatically generated

*Note.* Image obtained from *Practical Packet Analysis* (Sanders, 2017).

### Developing Signatures for Encrypted Traffic

In an article written by Papadogiannaki and Ioannidis (2021), the authors described the IDS system they had developed to account for traffic encryption. Papadogiannaki and Ioannidis stated that over 75% of current internet traffic is encrypted. Papadogiannaki and Ioannidis created an IDS solution named HeaderHunter. HeaderHunter accounted for encrypted traffic by only analyzing packet headers and metadata. Papadogiannaki and Ioannidis concluded that most IDS system utilizes rules that depend on deep packet inspection techniques. Encrypted traffic is impossible to analyze by deep packet inspection in the same fashion as unencrypted traffic (Papadogiannaki and Ioannidis, 2021).

In a blog post by Salesfore Engineering, John Althouse (2019) described JA3 and JA3S. JA3 is an open-source method that can be used to identify TLS clients within encrypted traffic, JA3S functions the same way but is used for the server-side of the TLS communications. At the time of writing, JA3 and JA3S could be obtained at https://github.com/salesforce/ja3. The OISF stated that Suricata comes with JA3 integration in the form of keywords such as ja3.hash and ja3.string. Both of these JA3 keywords have corresponding JA3S keywords (OISF, 2021).

## The Value of Rulesets in Signature-Based Network Security Monitoring Solutions

Broucek and Turner (2004) stated that signature-based IDS are only as valuable as the signature database. The OISF (2021) explained that Suricata uses rules that contain signatures to generate alerts. The rulesets are crucial to the functionality of the Suricata engine. Additionally, the OISF acknowledged that most Suricata administrators only utilize rulesets developed by rule writing entities like Emerging Threats (OISF, 2021).

In a document published by Proofpoint (2020) on their website, Proofpoint explained that their ET PRO ruleset contains rules for the latest vulnerabilities and malware campaigns. The industry standard ruleset is the Emerging Threats Pro ruleset developed and maintained by a division of Proofpoint known as the Emerging Threat team. Proofpoint reported that they analyze three million malware samples per day in their proprietary sandbox (Proofpoint, 2020).

### Default Rulesets

In an article entitled We Need to Talk About IDS Signatures. Shipulin (2018) explained that these currently available rulesets are the product of numerous hours of research and development by both individuals and entire organizations. Shipulin added that two of the most famous vendors who produce IDS rulesets are Emerging Threats and Cisco Talos. The organization Emerging Threats is part of a larger enterprise security company known as ProofPoint is responsible for developing a freely available set of IDS rules known as the ET Open ruleset and a paid alternative known as ET Pro. The ET Pro ruleset contains recently developed rules to detect active threats in a timely fashion (ProofPoint, 2021). The OISF stated that in most Suricata deployments, administrators utilize existing rulesets (2021).

## The Shortcomings of Signature-Based Network Security Monitoring

Andress (2019) stated that one of the most significant shortcomings of signature-based intrusion detection is that if a signature does not currently exist for a specific attack, it is likely that this will not generate an alert. Andress further explained that malicious actors might manipulate the traffic they generate specifically to avoid generating any alerts. These malicious actors may even utilize IDS tools in the process of testing their infrastructure (20l19). Vacca (2014) explained that signature-based IDS solutions depend on the signatures located within their rulesets to function. If an attack or an anomaly occurs, these signatures raise awareness by generating an alert. Vacca went on to explain that signatures that are too specific may lead to false negatives. Missed alerts are false negatives in the context of an IDS (Vacca, 2014).

Sanders and Smith (2014) explained that NSM is challenging to practice, and NSM practitioners require a specific skill set. To practice NSM effectively, an organization must employ personal with a specialized skill set to conduct the collection, detection, and analysis stages (Sanders & Smith, 2014). Bejtlich (2013) stated that network-based IDS cannot effectively monitor node to node traffic within the monitored network.

### False-Negative Events: Missed Alerts

Vacca, 2014 explained that false-negative alerts are events where an alert generation should have occurred but did not. In signature-based IDS systems, false-negative events can occur when an anomaly happens, but there is no signature to detect it (Vacca, 2014).

## The Importance of Rule-Writing

In the CIS Controls V8, the 13th control, Network Monitoring and Defense, states that we cannot rely on our network defenses never to fail. CIS further explained that security tools such as IDS are only effective when supported by a process of continuous monitoring (CIS, 2021a). Broucek and Turner (2004) explained that signature-based IDS solutions could not detect attacks or anomalies that do not yet have a signature developed for detection. Scarfone and Mell (2007) described a critical shortfall in many signatures that even the slightest variation may lead to a signature failing to report on the desired traffic anomaly.

Emerging Threats is an organization within Proofpoint, Inc. that develops rulesets for IDS systems such as Snort and Suricata. On Emerging Threats' (2019) website, they provided recommendations for IDS administrators on how to utilize rulesets. Emerging Threats recommended determining which rulesets are irrelevant to one's environment and disabling them. Additionally, Emerging Threats recommended writing rulesets that correspond to the Suricata administrator's network configuration. For example, analysts can write IDS rules can for the use of any ports which should be unused. The rules contained within both the Emerging Threats Open and Emerging Threats Pro rulesets had been developed by Emerging Threats to be used in most environments (Emerging Threats, 2013).

In a master's theses published by James Madison University, Rice (2014) had published a project automating the generation of signatures for the Snort IDS. Rice's goal was to develop a tool that generated Snort rules from any file type. Rice designed an algorithm that takes an input file path and outputs a syntactically acceptable rule for the Snort IDS. Rice stated that his project was able to simplify the rule development process while also allowing the process to be fully automated by utilizing a tool to periodically execute the rule generation script. Rice additionally acknowledged that his tool could assist in the rapid deployment of rules when a novel threat is located (Rice, 2014).

# Discussion of Findings

## The Value of Network Security Monitoring

In the simplest of terms, the practice of NSM can be used to detect bad events on a network. NSM involves the collection of network traffic and detection and analysis of anomalies within that traffic (Sanders & Smith, 2014) Network-based IDS can be used to detect anomalies within network traffic. Network-based IDS detect these anomalies by comparing packets of network traffic to signatures contained within the applied ruleset (Bejtlich, 2013). Signature-based IDS function similarly to signature-based AV solutions (Andress, 2019). The Suricata engine utilizes rulesets to identify the signatures which generate alerts. Additionally, practicing NSM provides entities with a valuable layer of defense.

The need for NSM is validated by the Center for Internet Security's CIS Controls V8. The 13th control from the Center for Internet Securities CIS Controls V8 is the ability to operate processes and tooling to practice network monitoring for security threats across an organization's network (CIS, 2021). A primary weakness of many security solutions is their inability to detect intrusions at a network level. Practicing NSM with a NSM solutions such as an IDS can account for this weakness.

## The Value Rulesets of Signature-Based Intrusion Detection Systems

In the official Suricata 7.0 documentation, the OISF stated that it is necessary to install signatures for Suricata as the engine utilizes those signatures to trigger alerts. The Suricata engine utilzes rulesets in order to generate alerts. The detection functionality within Suricata based IDS lies within the Suricata detection thread-modules. Suricata utilzes these thread modules in most available runmodes (2021). If no rulesets are applied, no IDS alerts will be generated.

### Signatures for Encrypted Traffic

Papadogiannaki and Ioannidis designed their HeaderHunter IDS to alert on encrypted traffic by focusing the signatures utilized by the IDS to the network packet header and metadata. HeaderHunter does not analyze the rest of the packet, which allows it to utilize fewer system resources (Papadogiannaki and Ioannidis, 2021).

Various projects have been developed which focus on performing detection on encrypted traffic. Engineers at Salesforce developed a solution known as JA3 and JA3S, which can be used to develop signatures for TLS encrypted traffic (Althouse, 2019). Suricata comes with JA3 and JA3S integrations. Suricata allows for matching on JA3 and JA3S strings and hash values (OISF, 2021).

## Signature-Based Intrusion Detection System Limitations

One factor that makes the rulesets in any Signature-based IDS systems are unable to detect threats for which they do not have a signature in their rulesets (Andress, 2019).

## Simplifying the Rule Writing Process

Tools such as the one developed within this research project provide Suricata analysts with more accessible routes to obtaining the rules they need. The development of future tools that automate steps of the rule-writing process will undoubtedly help NSM administrators develop and implement novel rules.

## Generating Suricata Signatures with Suri-rule-gen.py

The Python script, which was developed during this research project, was designed to output a Suricata rule which information from a command-line syntax. The author wrote the script for the Python 3 interpreter. The suri-rule-gen.py script is contained within a GitHub repository that can be located at https://github.com/CarrCyberSec/suri-rule-gen.

### Argparse Module Application

The Python module Argparse was used extensively to allow the user to set variables from a command-line interface syntax.

Text

Description automatically generated

### Input Validation

Input validation was performed in a verity of ways. For variables with a more finite number of valid values, lists were used to define the valid values, and then the variables was tested for being present in the list. For more complex values such as source IP and destination IP, regular expresssions had been developed. These regular expressions were designed to limit the invalid values entered while still being flexable enough to accept the various formats specified in the Suricata documentation.

<Screenshot of validation lists and regex>

<Screenshot of whileloop>

### File Handling

**Figure 6**

*File Handling from suri-rule-gen.py*

Text

Description automatically generated

## Future Research and Recommendations

Tools such as the one developed in this research project provide Suricata adminstrators with a lower bar of entry to the rule writing process. Additional tools can be further developed to assist Suricata adminsistrators such as a rule generators with a graphical user interface (GUI). Rule writing for intrusion detection systems and specifically Suricata can be further improved by developing additional tools which automating the development, testing, and implementation of these rules. Many organizations still lack the ability to rapidly deploy a novel signature.

Additionally, there are still few scholarly works related to the subject of NSM. There are numerous materials related to IDS and network traffic analysis, but few for NSM. While books and blog posts from subject matter experts and NSM practitioners provide many valuable insights, many individuals learning about NSM are negatively affected by the lack of quality reference materials (Sanders & Smith, 2014)

# Conclusion

This research project aimed to answer the following questions. The research contained within this project attempted to answer the questions: 1) Why is Network Security Monitoring (NSM) valuable? 2) How necessary are signatures in signature-based Network Security Monitoring (NSM) solutions, and 3) How can rule-writing for Suricata be simplified for Suricata administrators? Additionally, this research produced a Python script that can be used to generate rules that can be utilized by the Suricata engine. While this script can assist in the rule-writing process, poorly written rules can still have detrimental performance consequences to the Suricata engine. Therefore, all rules generated with suri-rule-gen should still have their quality verified before being implemented in a production environment.

The value of NSM comes from the ability for entities to detect and minimize the consequencs of intrsuions at a network level (Andress). NSM practiced using detection methods as prevention eventaully fails (Bejlitch, 2013; Sanders & Smith, 2014) All three steps of NSM must be practiced to effectivly locate anomolies at the network level.

# References

Alhomoud, A., Munir, R., Disso, J. P., Awan, I., & Al-Dhelaan, A. (2011). Performance Evaluation Study of Intrusion Detection Systems. *Procedia Computer Science*, 173-180.

Althouse, J. (2019). TLS Fingerprinting with JA3 and JA3S

https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967

Andress, J. (2019). Foundations of Information Security*.* No Starch Press.

Bejtlich, R. (2013). The Practice of Network Security Monitoring*.* No Starch Press.

Broucek, V., & Turner, P. (2004). Intrusion Detection: Issues and Challenges in Evidence Acquisition. *International Review of Law Computers*, 149-164.

Center for Internet Security. (2021a). *CIS Controls V8.*

https://www.cisecurity.org/controls/v8/

Center for Internet Security. (2021b). 4 Reasons SLTTs use Network Monitoring Systems.

*https://www.cisecurity.org/blog/4-reasons-sltts-use-network-monitoring-systems/*

Center for Internet Security. (2018). Albert: A Smart Solution for Network Monitoring.

https://www.cisecurity.org/blog/albert-a-smart-solution-for-network-monitoring/

Einay, S., Oz, C., & Navaei, Y. D. (2021). The Anomaly- and Signature-Based IDS for Network

Security Using Hybrid Inference Systems. *Mathematical Problems in Engineering*.

Emerging Threats. (2013) What Every IDS User Should Do.

https://doc.emergingthreats.net/bin/view/Main/WhatEveryIDSUserShouldDo

Emerging Threats. (2012) About Emerging Threats.

https://doc.emergingthreats.net/bin/view/Main/AboutEmergingThreats

Ganesan, A., Parameshwarappa, P., Peshave, A., Chen, Z., & Oates, T. (2019) Extending

Signature based Intrusion Detection Systems with Bayesian Abductive Reasoning https://arxiv.org/abs/1903.12101

IC3. (2021). *Internet Crime Report 2020:*

https://www.ic3.gov/Media/PDF/AnnualReport/2020\_IC3Report.pd*f*

OISF. (2021). Suricata User Guide

https://buildmedia.readthedocs.org/media/pdf/suricata/latest/suricata.pdf

Open Information Security Foundation & Applied Network Defense. (2017) Intrusion

Detection with Suricata [video] networkdefense.io.

https://www.networkdefense.io/library/intrusion-detection-with-suricata/

Papadogiannaki, E., & Ioannidis, S. (2021, February 6). Acceleration of Intrusion Detection in Encrypted Network Traffic Using Heterogeneous Hardware. *Sensors*.

Proofpoint. (2020) Proofpoint ET Pro Ruleset.

https://www.proofpoint.com/sites/default/files/pfpt-us-ds-et-pro-ruleset.pdf

Rice, B. (2014). Automated Snort Signature Generation [Master's Theses, James Madison

University]. JMU Scholarly Commons. https://commons.lib.jmu.edu/cgi/viewcontent.cgi?article=1314&context=master201019

Sanders, C. (2017). Practical Packet Analysis (3rd ed.). No Starch Press

Sanders, C., & Smith, J. (2014) Applied Network Security Monitoring. Syngress

Scarfone, K., & Mell, P. (2007, February). *NIST SP 800-94.* Fhttps://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf

Shipulin, K. (2018). We need to talk about IDS signatures. *Network Security*(3), 8-13. https://doi.org/10.1016/S1353-4858(18)30024-2

Sikorski, M., & Honig, A. (2012). Practical Malware Analysis. No Starch Press.

Vacca, J. R. (2014). Network and Systems Security (2nd ed). Syngress.

https://doi.org/10.1016/C2011-0-08779-3

# Appendix A

suri-rule-gen.py Source Code

#!/usr/bin/env python3

import os

import argparse

import re

import logging

#output file

outfile = 'suri-rule-gen.rules'

# Header Variables

rule\_action = "alert"

protocol = "ip"

source\_ip = "any"

source\_port = "any"

direction = "->"

dest\_ip = "any"

dest\_port = "any"

#option\_variables

message= "!!!Describe The Rule Here!!!"

rev='rev:001'

sid='sid:000001'

list\_of\_vars\_in\_options = [rev,sid]

#logging configuration

logging.basicConfig(filename='suri-rule-gen.log', filemode='a+', format='%(asctime)s-%(levelname)s-%(message)s', datefmt='%Y-%m-%d %H:%M:%S')

generated\_rule\_sid = 'Generated rule sid' + sid

#Regular Expressions for input validation

ip\_pattern = re.compile("^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$|^[\!]\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$|^\$HOME\_NET|^\$EXTERNAL\_NET|^\[|any|^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\/\d{1,3}$|^\$EXT\_NET")

port\_pattern = re.compile("^\d{1,6}|^\[\d{1,6}|^any|^!d{1,6}|^!\d{1,6}")

# Lists for Input Validation

action\_options = ['alert', 'pass', 'drop', 'reject', 'rejectsrc', 'rejectdst', 'rejectboth']

proto\_options = ['tcp', 'udp', 'icmp', 'ip', 'http', 'ftp', 'tls', 'smb', 'dns', 'dcerpc',

'ssh', 'smtp', 'imap', 'modbus', 'dnp3', 'enip', 'nfs', 'ike', 'krb5', 'ntp'

'dhcp', 'rfb', 'rdp', 'snmp', 'tftp', 'sip', 'http2']

direction\_options = ['->','<>']

# CLI arguments

parser =argparse.ArgumentParser()

parser.add\_argument('--action', action="store", type=str, help="Use to set rule action - Default is alert")

parser.add\_argument('--protocol', action="store", type=str, help="Use to set protocol.")

parser.add\_argument('--sip', action="store", type=str, nargs='+', help="Use to set source IP for rule. Format examples: | 10.0.0.0 | 10.0.0.0/8 | !10.0.0.0 | [10.0.0.0, 192.168.0.0/24, !172.16.0.0]")

parser.add\_argument('--srcport', action="store", type=str, nargs='+', help="Use to set the source port. Format exampels: | 80 | [80,81,82] | [8080:] | !80 | [1:80,![2,4]]")

parser.add\_argument('--direction', action='store', type=str, help="Use to se the direction of the rule valid options inclde: ")

parser.add\_argument('--dip', action="store", type=str, nargs='+', help="Use to set destination IP for rule. Format examples: | 10.0.0.0 | 10.0.0.0/8 | !10.0.0.0 |[10.0.0.0, 192.168.0.0/24, !172.16.0.0]")

parser.add\_argument('--destport', action="store", type=str, nargs='+', help="Use to set the destination port. Format exampels: 80 | [80,81,82] | [8080:] | !80 | [1:80,![2,4]]")

parser.add\_argument('--message', action="store", type=str, nargs='+', help="Use to set a descriptive message about the rule.")

parser.add\_argument('--meta', action="store", nargs='+', type=str, help="Used to set metadata variables. Be careful with formatting. sample format: --meta key value | --meta key value, key value")

parser.add\_argument('--ttl', action="store", type=str, help="Use to set TTL value. Format: number")

parser.add\_argument('--outfile', action="store", type=str, help="used to specify a file to use instead of suri-rule-gen.rules. MUST END IN .rules" )

parser.add\_argument('--rev', action="store", type=str, help="Use to specify Revision Number")

parser.add\_argument('--sid', action="store", type=str, help="Use to specify Signature Identification.")

parser.add\_argument('--content', action="store", type=str, nargs='+', help="Used to specificy payload content.")

parser.add\_argument('--classtype', action="store", type=str, help="Used to set classtype")

parser.add\_argument('--urlref', action="store", type=str, help="Used to set URL reference. Format: format.com")

parser.add\_argument('--cveref', action="store", type=str, help="Use to set CVE reference. Format: CVE-2021-1234")

parser.add\_argument('--priority', action="store", type=str, help="Use to set the rule priorty. Format: 1")

parser.add\_argument('--ja3', action="store", type=str, help="Use to set JA3 hash value, JA3 strings require md5 hashing.")

parser.add\_argument('--ja3s', action="store", type=str, help="Use to set JA3S hash value, JA3S strings require md5 hashing.")

#turn cli args into arg.<argument>

args = parser.parse\_args()

#While loops to test the presence of CLI arguements

while True:

if args.action is not None:

if args.action.lower() in action\_options:

rule\_action = args.action.lower()

logging.info('Generated rule with sid:' + sid +'Rule action set: ' + rule\_action)

break

else:

print("invalid selection")

logging.error(generated\_rule\_sid + ' invalid action entered: ' + args.action)

break

else:

#print('no action selected')

logging.info(generated\_rule\_sid+' no alert value entered.')

break

while True:

if args.protocol is not None:

if args.protocol.lower() in proto\_options:

protocol = args.protocol

logging.info(generated\_rule\_sid +'Rule protocol set: ' + protocol)

break

else:

print('invalid protocol selected valid protocol\n' + str(proto\_options))

logging.error(generated\_rule\_sid + ' invalid protocol entered: ' + args.protcol )

break

else:

print('no protocol selected')

logging.info(generated\_rule\_sid + ' no protocol value entered.')

break

while True:

if args.sip is not None:

test\_source\_ip = ' '.join(args.sip)

if ip\_pattern.match(test\_source\_ip) is not None:

source\_ip = test\_source\_ip

break

else:

logging.log(logging.ERROR,'Invalid Source IP entered: ' + test\_source\_ip)

print('!!!!!Invalid Source IP entered!!!!!')

logging.error('Genereated rule sid:' + sid + ' invalid source IP entered: ' )

break

else:

print('no Source IP specified with --sip')

break

while True:

if args.srcport is not None:

test\_source\_port = ' '.join(args.srcport)

if port\_pattern.match(test\_source\_port):

source\_port = test\_source\_port

logging.info('Generated rule sid:' + sid + ' source port value:' + source\_port)

break

else:

print("The port was not entered in the correct format.")

logging.error('Generated rule sid:' + sid + ' invalid source port entered')

break

else:

print('no Source port was specified with --srcport')

logging.info('Generated rule sid:' + sid + 'no value source port value entered.')

break

while True:

if args.direction is not None:

if args.direction in direction\_options:

direction = args.direction

logging.info('Gerented rule sid;')

break

else:

print('invalid direction selected: please use \n \<\> or \-\> when using a bash terminal')

logging.error('Generated rulue sid:' + sid + ' invalid rule direction ' + args.direction )

break

else:

print('no direction specified.')

logging.info('Generated rule sid:' + sid + 'no rule driection specified.')

break

while True:

if args.dip is not None:

test\_dest\_ip = ' '.join(args.dip)

if port\_pattern.match(test\_dest\_ip):

dest\_ip = test\_dest\_ip

logging.info('Generated rule sid:' + sid + '')

break

else:

print('The destination IP was not in the correct format.')

break

else:

print('No Dest IP specified.')

break

while True:

if args.destport is not None:

test\_dest\_port = ' '.join(args.destport)

if port\_pattern.match(test\_dest\_port):

dest\_port = test\_dest\_port

logging.info('Generated rule sid:' + sid + 'dest port set to:' + dest\_port)

break

else:

logging.error(logging.info('Generated rule sid:' + sid + 'dest port entered incorrectly: ' + dest\_port))

print('Destionation port entered incorrectly.')

break

else:

print('no dest port specified')

logging.info('Generated rule sid:' + sid + 'no dest port specified specified.')

break

#Rule options start here

while True:

if args.message is not None:

message = " ".join(args.message)

break

else:

print("warning: no message specified. Please use --message to specify the rule message")

break

while True:

if args.sid is not None:

sid = 'sid:'+args.sid

break

else:

break

while True:

if args.rev is not None:

rev = 'rev:'+args.rev

break

else:

break

while True:

if args.meta is not None:

meta\_var\_constructor = " ".join(args.meta)

meta\_var = 'metadata: ' + meta\_var\_constructor

logging.info('Generated rule sid:' + sid + ' meta var set to: ' + meta\_var )

list\_of\_vars\_in\_options.insert(1, meta\_var)

break

else:

break

while True:

if args.ttl is not None:

ttl = 'ttl:' + args.ttl

list\_of\_vars\_in\_options.insert(1, ttl)

break

else:

break

while True:

if args.content is not None:

content\_constructor = ' '.join(args.content)

content = 'content;' + content\_constructor

list\_of\_vars\_in\_options.insert(1, content)

logging.info('Generated rule sid: ' + sid + 'content set to: ' + content)

break

else:

break

while True:

if args.classtype is not None:

classtype = 'classtype:'+args.classtype

list\_of\_vars\_in\_options.insert(1, classtype)

logging.log(1,'Generated rule sid:'+ sid + 'classtype set to:' + classtype)

break

else:

break

while True:

if args.urlref is not None:

ref = 'reference: url, ' + args.urlref

list\_of\_vars\_in\_options.insert(1, ref)

logging.log(1, 'Generated rule sid:' + sid + 'refernce set to: ' + ref)

break

else:

break

while True:

if args.cveref is not None:

ref = 'reference: cve, ' + args.cveref

list\_of\_vars\_in\_options.insert(1, ref)

break

else:

break

while True:

if args.priority is not None:

priority = 'priority:'+args.priority

list\_of\_vars\_in\_options.insert(1, priority)

logging.log(1, 'Generated rule sid:'+ sid + 'priority set to: ' + priority)

break

else:

break

while True:

if args.ja3 is not None:

ja3 = 'ja3.hash; content:' + args.ja3

list\_of\_vars\_in\_options.insert(1, ja3)

logging.log(level=1, msg='Generated rule sid:' + sid + 'ja3 value set to: ' + ja3)

break

else:

break

while True:

if args.ja3s is not None:

ja3s = 'ja3s.hash; content:' + args.ja3s

list\_of\_vars\_in\_options.insert(1, ja3s)

logging.log(level=1, msg='Generated rule sid:' + sid + 'ja3s value set to: ' + ja3s)

break

else:

break

#Test if a different outfile should be used

while True:

if args.outfile is not None:

outfile = args.outfile

logging.info('Generated rule sid:' + sid + 'outfile set to: ' + outfile)

break

else:

break

#CONSTRUCTING THE RULE

#message prefix should be present in all rules.

list\_of\_vars\_in\_header = [ rule\_action, protocol, source\_ip, source\_port, direction, dest\_ip, dest\_port]

message\_constructor = ' (msg:"' + message + '"'

list\_of\_vars\_in\_options.insert(0,message\_constructor)

new\_rule\_header = ' '.join(list\_of\_vars\_in\_header)

new\_rule\_options = '; '.join(list\_of\_vars\_in\_options) + ';)'

new\_rule = new\_rule\_header+new\_rule\_options

print(new\_rule)

#Check if outfile exists and create or write to it

if os.path.exists(outfile):

f = open(outfile, "a")

f.write(new\_rule + '\n')

else:

f = open(outfile, "x")

f.writelines("%s\n" % new\_rule)