ARL-SR-0485 ● DEC 2023

DEVCOM ARMY RESEARCH LABORATORY

Hands-on Cybersecurity Studies: Automated Scan Tool Detection

by Jaime C Acosta

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-SR-0485 ● DEC 2023

Hands-on Cybersecurity Studies: Automated Scan Tool Detection

Jaime C Acosta
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
December 2023	Special Report	2/01/2023	9/30/2023

4. TITLE AND SUBTITLE
Hands-on Cybersecurity Studies: Automated Scan Tool Detection

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Jaime C Acosta

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
DEVCOM Army Research Laboratory ATTN: FCDD-RLA-ND Adelphi, MD 20783	ARL-SR-0485

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID: Jaime C Acosta, 0000-0003-2555-9989

14. ABSTRACT
This special report describes a hands-on cybersecurity exercise that focuses on novel research aimed at automatically collecting data and generating intrusion detection system (IDS) rules using the Generate, Examine, Match (GEM) system. In the exercise, participants create a network scenario and learn how to collect and analyze the traffic generated by a network scanning tool. Afterward, they create an IDS rule by hand. The subsequent steps lead students through how to use the GEM tool and then they automatically create IDS rules.

15. SUBJECT TERMS
cybersecurity; scan tools; intrusion detection system; Collaborative Innovation Testbed; CyberRIG; Network, Cyber, and Computational Sciences

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	37

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Jaime C Acosta	(575) 993-2375

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction

Network scanning tools are used to collect information such as live hosts and the services that they are running. Many of these tools are available publicly and can be used by both administrators and adversaries. For this reason, it is critical that defenders can identify the use of these tools on networks. This can help determine intent and mitigations such as network hardening and honeypot deployment. However, many of these scan tools do not have readily available detection signatures. This is often due to the need for subject-matter experts to learn how to use the tools, create a viable network environment, capture network data, and then hand derive the signatures. Novel research in the field has shown that much of this process can be automated, and it can be effective.[1] This report describes the Generate, Examine, Match (GEM) software infrastructure, which allows analysts to create scan tool signatures by simply defining a configuration file that describes the paths and execution arguments for the tools. GEM uses text similarity algorithms to extract patterns and converts the patterns into detection rules.

1.1 Exercise Overview

In this exercise, participants will learn how to use the GEM tool by first conducting the signature creation process manually. This involves creating a network scenario using the common open research emulator (CORE),[2] setting up a scanner node and a node to scan. Afterward, the scenario is executed, data is collected, and then a signature, based on the network traffic, is handwritten and tested with the Suricata[3] intrusion detection system (IDS). In the next part of the exercise, participants create a configuration file for the Commix[4] network scan tool. This involves creating and populating a Java Script Object Notation (JSON)[5] formatted file, including the path to the executable and the ports that are used by the tool. Afterward, the GEM system generates several files for two separate scenarios that use the Commix tool. Network captures, longest common substrings across the execution, and intrusion detection rules are created automatically. The exercise concludes when participants rerun the rules with Suricata and observe the alerts that trigger with Commix scans.

1.2 Open Source Scanning Tools

Many scanning tools are widely and freely available. In the Kali Linux Operating System alone,[6] there are over 40 stand-alone scanning tools as well as over 1000 auxiliary scanners included with the Metasploit framework.[7] These tools facilitate the process of identifying live machines and their services. Some tools are passive meaning that they do not actively send packets to specific machines, but instead

simply listen to traffic and infer information from the data (i.e., live devices from source and destination traffic information). Active tools send packets to network addresses, read responses, and in many cases are able to detect specific services and versions. While many tools have common objectives, they contain nuances that may help system administrators understand an adversary's intent.[1]

As an example, Kali Linux documentation describes most of its tools. These descriptions can indicate the intended use of a specific tool. The masscan tool[8] is similar to the Nmap tool[9] in that they both have the ability to scan for many commonly used services. However, according to their documentation, masscan is for large-scale scanning (used to scan the entire Internet) as it has the ability to conduct distributed scanning; Nmap is typically used at smaller scales.

1.3 The Generate, Examine, Match System

The GEM system is built on several open source tools. It is written in the Python[10] programming language and generates and executes scenarios using CORE. Scan tools are incorporated into GEM as services (which run on CORE nodes) and/or external artifacts (virtual machines [VMs], containers, and/or physical hardware). The system uses a plug-in architecture, which allows analysts to swap out certain parts of the system, including the text similarity analysis portion, which by default uses the Difflib Python module[11] to extract patterns across multiple executions. The output is by default Suricata intrusion detection rules, but this can also be swapped out to create Python code snippets, XML files, and others.

2. Setup and Configuration

The setup of the exercise includes a VM with the following software:

- Oracle VirtualBox 6.1.30 64-bit[12]
- Kali Linux 2022.1 64-bit VM
- CORE 8.2.0
- Visual Studio Code 1.56.1[13]

The US Army Combat Capabilities Development Command Army Research Laboratory South Cyber Rapid Innovation Group (CyberRIG) Collaborative Innovation Testbed (CIT)[14] is used to host the Kali Linux VM. The VM has CORE preinstalled for network scenario creation and traffic capture. Visual Studio Code is used to create and populate configuration files and to view textual data. Configurations required by the participant for the Kali VM are explained in the exercise.

3. Learning Objectives

The general purpose of the exercise is to provide participants with network scanning tools as well as the process of developing IDS signatures.

More specifically, the learning objectives associated with the exercise are as follows:

- **Scan Tools.** This exercise enables participants to gain a hands-on understanding of what network scanning tools are, specifically focusing on the Commix network scanner, by analyzing its execution, output, and the traffic it generates.
- **Network Scenario Creation.** An introduction to the popular and open source CORE tool that allows users to efficiently create network scenarios with a graphical interface. Nodes are configured during the exercise to run tools and collect data.
- **IDS Signatures.** Participants will learn the basic structure of an IDS rule. They will then apply the knowledge they gain from network traffic analysis to create a rule for the Commix tool.
- **Automatic Rule Generation with GEM.** After manually creating a signature, participants will learn how to use the GEM tool to automate the process of rule generation. They can also experiment with other scanning tools at the end of the exercise.

4. Exercise

The following exercise is intended to provide participants with a comprehensive walk-through, comprising a series of step-by-step tasks. Each task will be conducted within the Kali Linux VM. Note that all data and systems involved in this exercise are entirely fictional and simulated and are exclusively for educational purposes.

Participants are given the following information at the start of the exercise:

> Network scanning tools are used to identify devices and services. These tools are often used by network administrators and security personnel for testing and verification, but they can also be used by adversaries. Intrusion detection systems (IDS) are used to identify these tools; however, methods to generate detection rules automatically are lacking.

In this exercise, you will learn about network scanning tools and become familiar with the traffic they generate. You will also learn to use the Generate, Examine, Match (GEM) tool to automate the process of collecting scanner traffic and creating IDS rules.

4.1 Step 1: Create a Network Scenario and Collect Traffic

****

The Common Open Research Emulator (CORE) is software that allows the creation of networks all within a single computer using a graphical user interface (GUI). Both simple and complex networks can be created with ease using the concept of nodes, which can be thought of as individual devices (including computers, routers, switches, etc.).

CORE uses a client/server architecture, meaning that you can run one-to-many scenarios with or without a GUI. The bulk of the processing is handled by the CORE service (called CORE-daemon).

****

You should have been given connection information for the Collaborative Innovation Testbed (CIT). If you have not logged in to CIT through your browser, please do so now.

1) If your screen is blank, hit enter and then login using the following credentials:
    - username:**kali**
    - password:**kali**
2) Open a new terminal by clicking on the Terminal icon on the top left side of the screen: 
3) Start the ***CORE*** service or the CORE-daemon by running the following command:
    - **sudo core-daemon**

      *When prompted for a password, enter **kali***

      You should see something similar to Fig. 1.

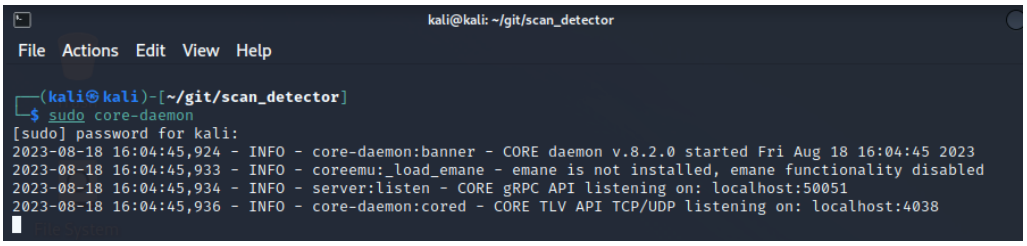

Fig. 1 Start the CORE-daemon

The CORE service is now running with administrator privileges, which are required for all of its networking (i.e., creating bridges, updating routes, and managing firewalls). It is listening for requests for processing. Next, you will connect to the service through the GUI. You will notice several commands and messages in this terminal. These are a prime source for troubleshooting and debugging, but you can ignore these for now.

4) Start another terminal by again clicking on the Terminal icon on the left side of the screen.

5) Run start the CORE GUI by running the following command:

   core-gui-legacy

6) You should now see the main CORE interface as shown in Fig. 2.

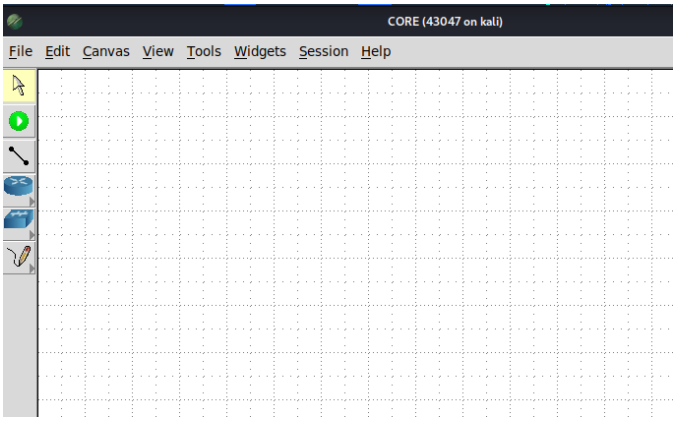

Fig. 2 CORE canvas

7) Create a simple network that consists of two “laptop” nodes connected to each other.

   a. Click on the PC nodes as shown in Fig. 3.

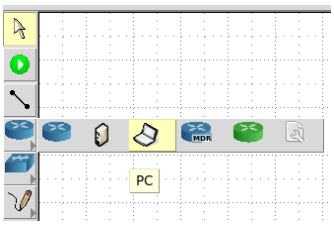

Fig. 3 CORE PC node

b. Add two of the nodes by clicking twice on the CORE canvas as shown in Fig. 4.

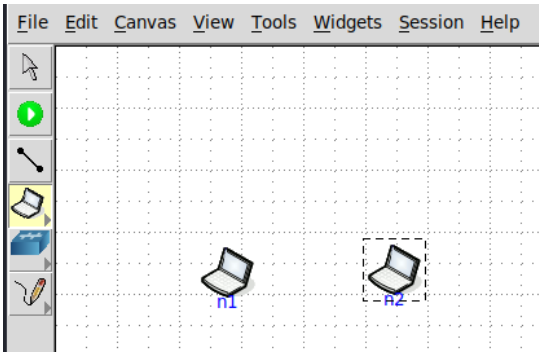

Fig. 4 CORE add two nodes

c. Connect the nodes together by selecting the link tool and then clicking on **n1** and dragging to **n2** as shown in Fig. 5.

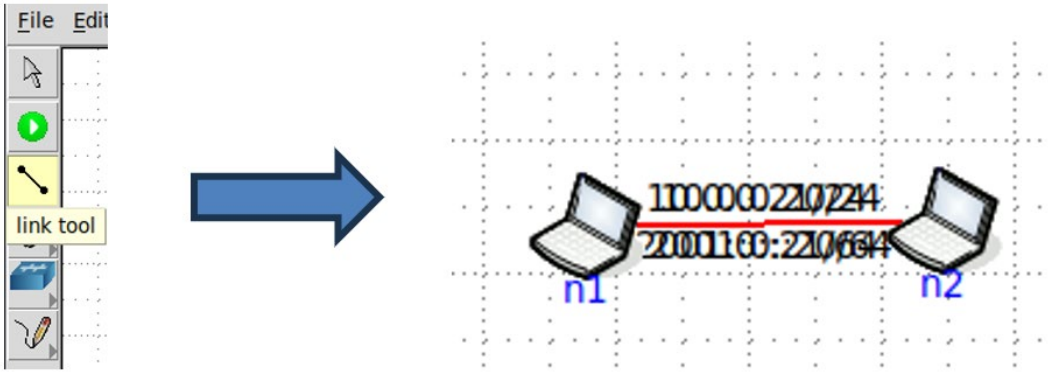

Fig. 5 CORE link tool

d. Click on the selection tool on the left panel and then move the nodes around to help with visibility as needed (Fig. 6).

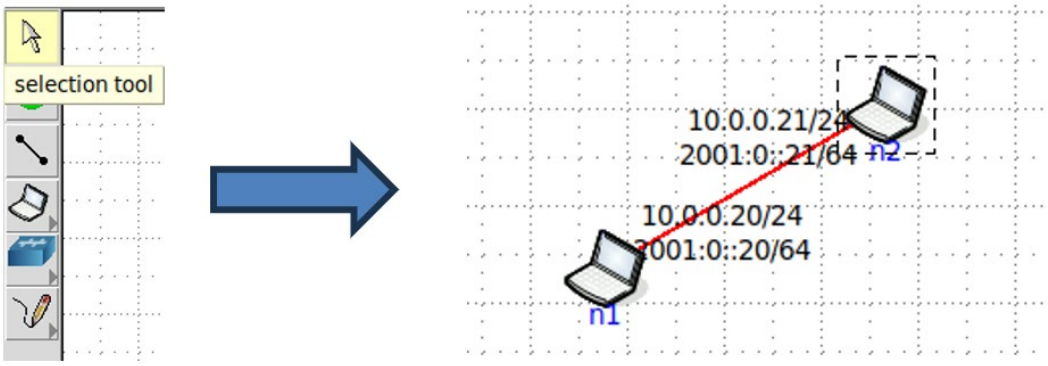

Fig. 6 CORE better node view

Notice that the two nodes have number labels associated with them. Specifically focus on the upper labels that take the form: xx.x.x.xx/xx. These are the IPv4 addresses that the nodes will use when you start the scenario or emulation.

8) Start the scenario/emulation by clicking on the play (start the session) button (Fig. 7).

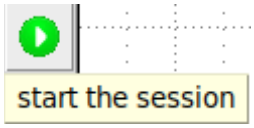

Fig. 7 Start button

Once you press the button, it will turn into a "stop the session" button. This means that the emulation is running.

****

At this point, the nodes in the scenario are running and can be used to execute commands just like a real system. CORE uses kernel namespaces to provide each of the nodes its own executing copy of the networking stack, among many other functions. It also provides each node with private file systems/folders. However, each of these nodes is a sort of copy of the host operating system (in this case Kali). A nice benefit of this is that each node can execute almost any command/tool available on the host. You will start by running the Ping command.

The Ping command is a well-known networking tool used to determine whether devices are "up" and reachable. It can also be used to determine latency associated with the communication.

****

9) Double-click on the **n1** node and a terminal will open. You can run commands on this terminal and they will be executed in the context of this node.

10) From this newly opened terminal, run the ping command as shown in Fig. 8 to test whether you can "reach" the other node in the emulation.

- **ping -c 4 10.0.0.21**

If successful, the terminal should be similar to Fig. 8.

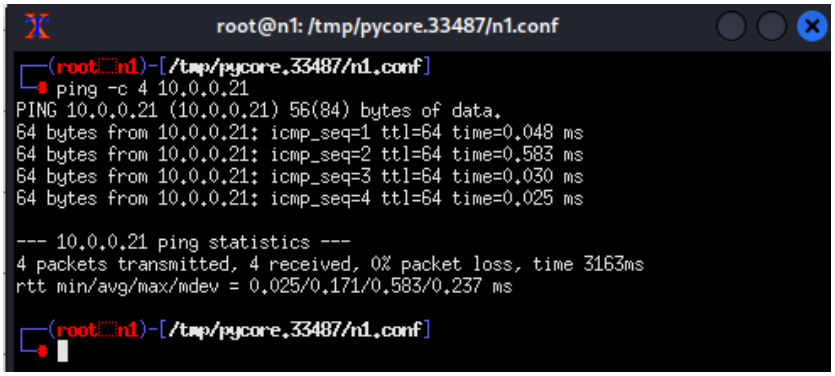

Fig. 8 Ping output

11) Stop the emulation by clicking the stop button  located on the left panel.

****

CORE allows users to configure services that should run on nodes once the emulation starts. In the next few steps, you will set up a node to start capturing traffic using a tool called "dumpcap."

The network sniffing tool dumpcap is capable of capturing and logging all network traffic that arrives at a node.

****

12) Right-click on the **n2** node, click the **Configure** label, then **Services**, and finally on the **wrench button** next to **UserDefined** (Fig. 9).

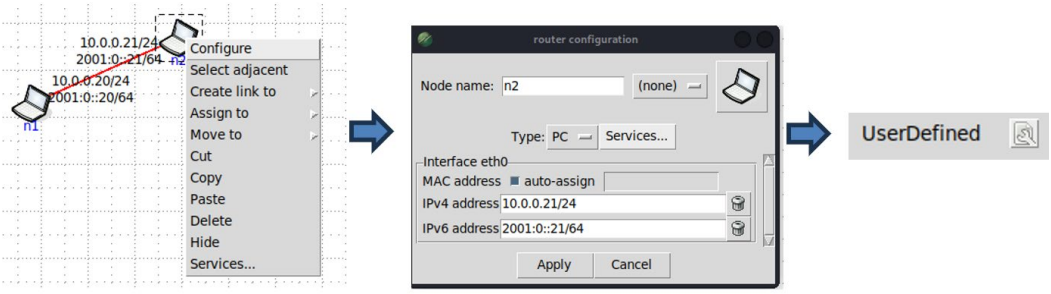

Fig. 9 CORE UserDefined service

13) Fill in the fields in the window as shown in Fig. 10 and **afterward** click on the **paper icon** (***DO NOT press the Apply button yet***). Notice that you are saving the network capture to a file called **/tmp/mycap01.pcap.**

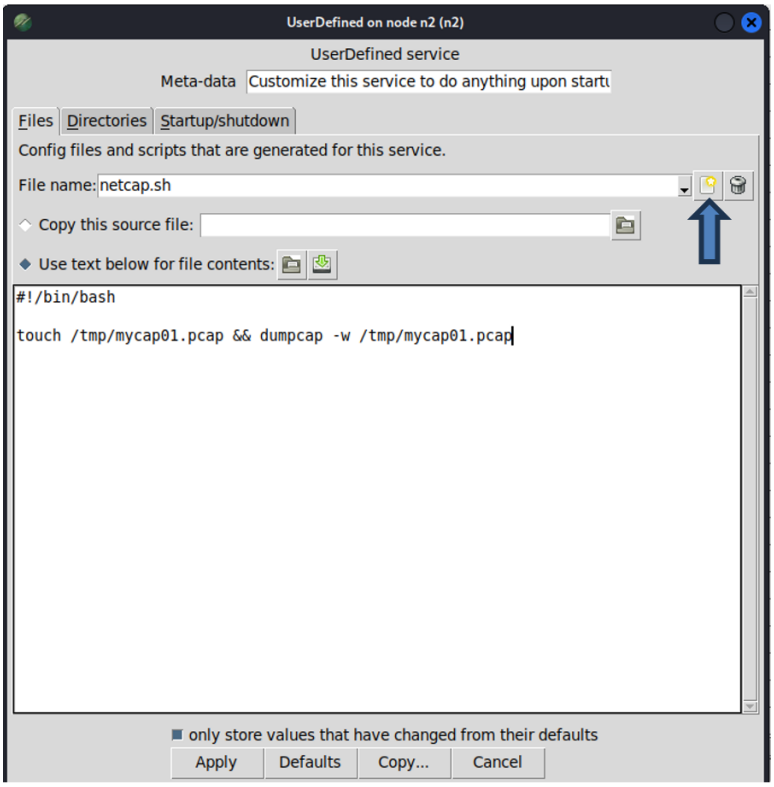

Fig. 10 CORE script input

14) Click on the **Startup/shutdown** tab and enter the following in the **Startup Commands** entry field and then click on the **paper icon** (Fig. 11):

- **/bin/bash netcap.sh**

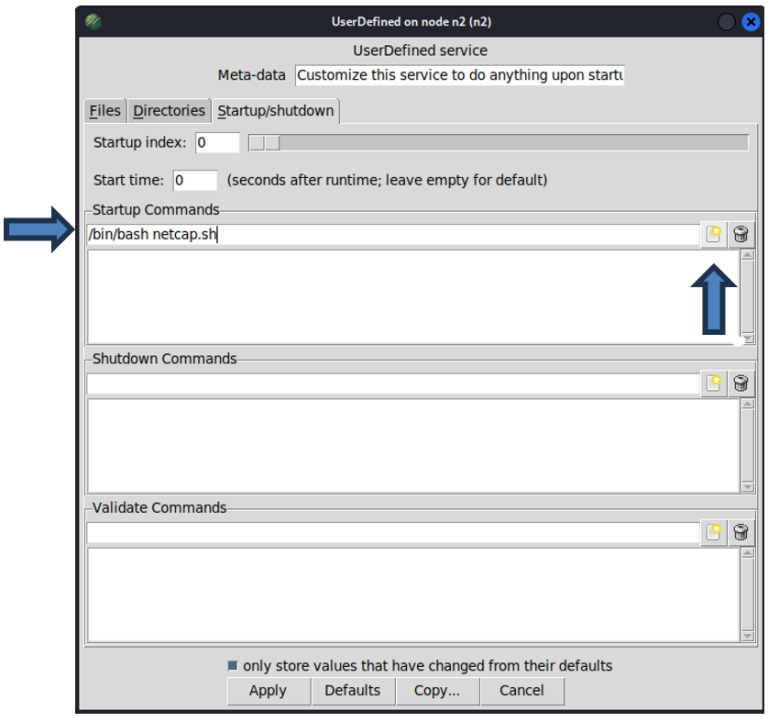

Fig. 11 CORE script startup

15) Ensure that your window matches Fig. 12 and then click the **Apply** button and all of the remaining pop-up windows.

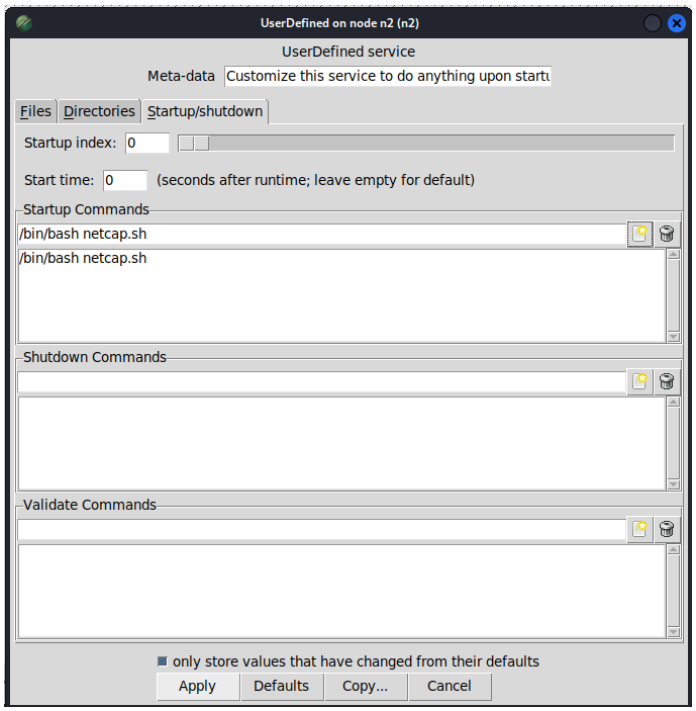

Fig. 12 CORE capture on startup

16) Save the current CORE scenario by clicking on **Files>Save As imn…** Save your file to **/home/kali/Desktop** and name it **myscen01.imn** (Fig. 13).

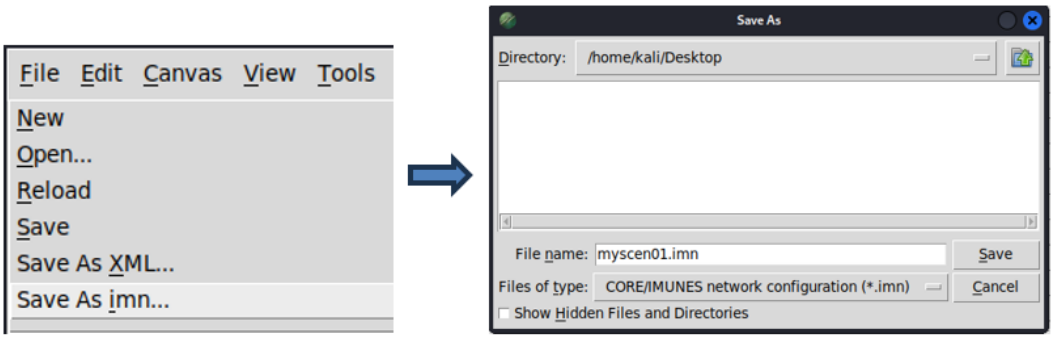

Fig. 13 Save myscen01.imn

17) Save your file on the desktop and name it **myscen01.imn**.

18) Start the emulation by pressing the start button (Fig. 14).

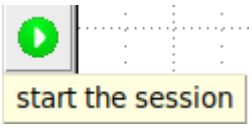

Fig. 14 CORE start button

19) Double-click on the **n1** node and once again ping the **n2** node (Fig. 15).

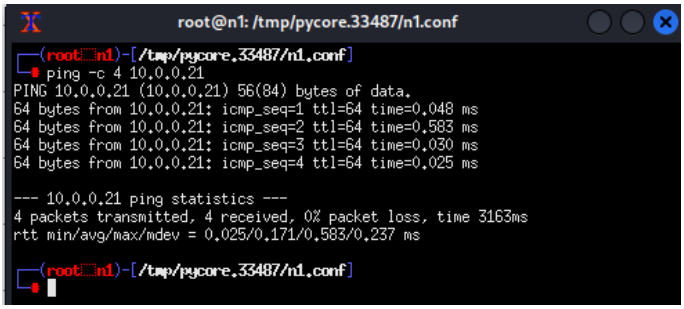

Fig. 15 Second ping output

****

The Kali Linux Operating System comes prepackaged with many network scanning tools. These are used to identify services that are actively running on devices. Many of these scanning tools use a predefined set of network packets to certain "ports" (think of it as a certain ID used by specific services) to determine the status of the services.

The Commix tool is used to test whether services contain bugs, errors, and vulnerabilities. To observe the network traffic that Commix uses to identify a web server during a scan, you will need to start a listener (using a tool called **ncat**) on the remote machine before running the scan. Note that this is not a real web server, but instead a generic process that only listens for connections and prints out to the terminal any data it receives.

Since the Commix tool targets web applications, you will use the "port" that is commonly used by web applications: **80**. You will complete this in the following steps.

****

20) Double-click on the **n2** node and run a simple network listener using the **ncat** tool.
    - **ncat -l -p 80**
21) Back in the terminal for node **n1**, use the Commix scanner against the remote node by running the following command (ensure that you are running this in the **n1** terminal).
    - **commix -u http://10.0.0.21**
22) If you did everything correctly, you should see something that looks like Fig. 16 in the **n2** terminal window.

Fig. 16 Commix terminal output

This window shows you the string that the Commix tool sent over the network. If a real web server was running on **n2**, it would have responded with its default web page and Commix would then send additional packets to determine more information about the service.

Focus on the **User-Agent** string. Notice that it contains the name and version of the tool: **commix/v3.3-stable**. This is valuable information that we can use to identify the scanning tool.

23) Stop the emulation by clicking on the stop button: 

24) At this time you can close the CORE GUI by clicking on the **X** in the top right corner (Fig. 17).

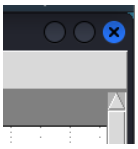

Fig. 17 CORE close window

****

Networked systems commonly have intermediary nodes that capture network traffic and "alert" on suspicious traffic using an IDS with a set of predefined rules. In this case, the rule could look for the string **commix/v3.3-stable** to identify and alert on the Commix traffic.

Network traffic captures can be used to analyze and create such rules. In the next steps, you will use a well-known graphical traffic analyzer called Wireshark to view the network traffic associated with the exchange in the previous steps and ultimately create an IDS rule to identify the Commix scan.

****

25) Recall that you created a **UserDefined** service on **n2** to capture network traffic. Open a new terminal by clicking on the Terminal icon on the top left side of the screen  and run the following command:

- **sudo wireshark**

  When prompted for a password, enter **kali**.

Open the network capture that was collected from the **n2** in the CORE emulation by clicking on **File>Open** and selecting **/tmp/mycap01.pcap**.

****

Wireshark shows the network packets using several views. Figure 18 describes some of the main parts.

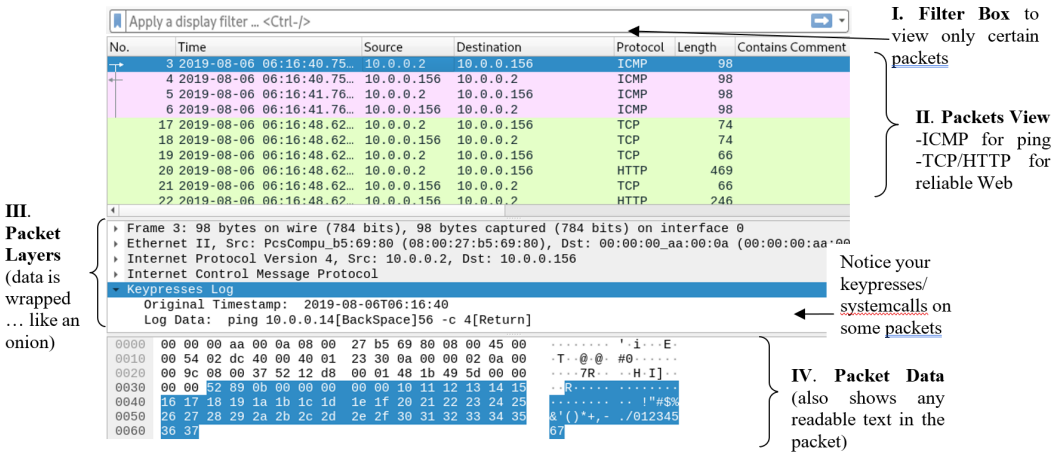

Fig. 18 Wireshark window explanation

****

26) Scroll down the **Packets View** (labeled **II** in Fig. 18) until you find the first packet with **ICMP** (NOT ICMPv6) under the **Protocol** column. Look at the **Info** column and notice the request/reply pairs. This is the traffic associated with the ping command that you ran during the emulation. Write down the Source and Destination values of any of the **ICMP** packets here:

    Source: _____ Destination: _____

27) Scroll down the **Packets View** until you see a packet that has **HTTP** under the **Protocol** column.

28) ***Packet View Information***: To learn more about this packet, in the **Packets View**, fill in the following information about this packet.

    - At what time was it sent? (round to the nearest tenth of a second) _____
    - To what IP address was it sent (destination)? _____
    - How many bytes long is the packet (length)? _____

29) ***Packet Layer Information***: Network packets consist of **Layers** of data (kind of like a mailed letter has paper inside an envelope and address information on the envelope). HTTP is layered on top of something called the **Transmission Control Protocol** (**TCP**) layer.

30) Look at the **Packet Layers** (**III** in Fig. 18) for your **HTTP** packet and look at the **TCP** layer to answer these questions:

- What is the **source port** number? _____
- What is the **destination port** number? _____

31) ***Packet Data***: In the **Packet Layers**, click on the label that reads **Hypertext Transport Protocol**. Now look at the **Packet Data** (**IV** in Fig. 18).

The associated packet data is highlighted in blue as shown in Fig. 19.

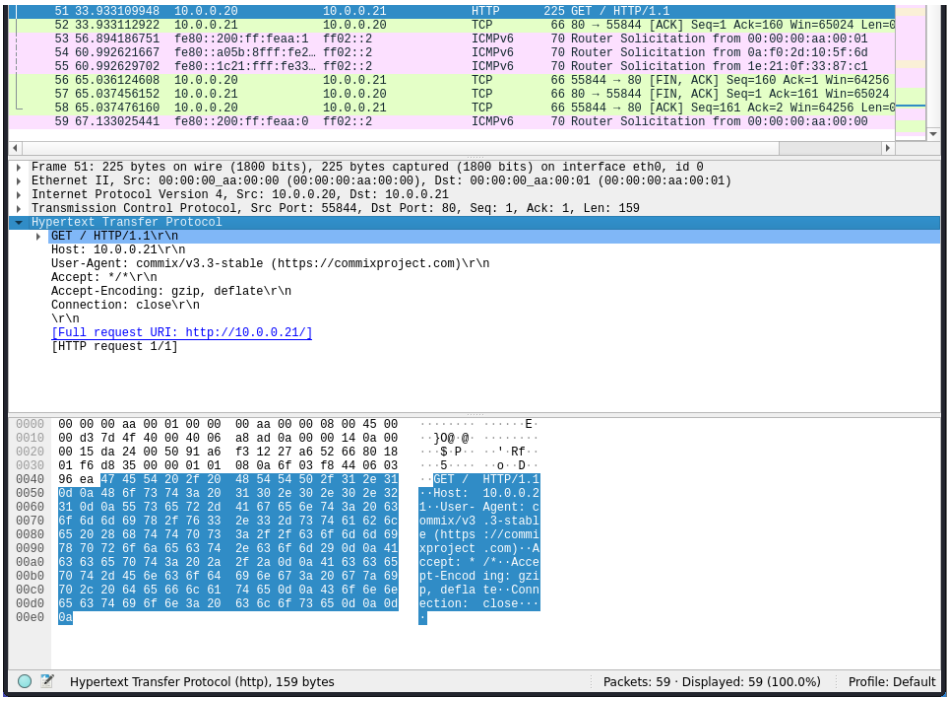

Fig. 19 Commix packet in Wireshark

The highlighted data should look familiar. This is what was shown in the **n2** terminal windows after the Commix tool was executed.

Close all open windows, including **Wireshark** and any **terminals**.

You have now completed part 1 of this exercise. To summarize, you have done the following:

- Created a simple CORE scenario consisting of two network nodes.
- Experimented with the ping command, which is used to identify live nodes on a network.
- Executed a simple network scan using the Commix tool.
- Logged and analyzed the traffic generated by Commix using the Wireshark tool.

4.2 Step 2: Create an IDS Rule

****

When creating IDS rules, it is important that they are not too broad or specific, which may lead to an overwhelming amount of alerts or not identifying critical behavior. Thus, it is important to identify patterns that are accurate and persistent. For example, in Section 4.1 the string **commix/v3.3-stable** was present as were others such as **HOST: 10.0.0.21**.

In this section you will look at two network captures that contain Commix scan data: 1) the pcap that you collected in part 1 and 2) another pcap that was collected outside of this exercise on a separate network using different source and destination addresses. You will then generate and test an IDS rule based on the similarities that you observe across these two captures.

****

1) Open a terminal window and open two instances of Wireshark. The first will open the capture that you collected in part 1 (located in **/tmp/mycap01.pcap**). The second will open a previously captured file (located in **/home/kali/data/cap00.pcap**) that contains commix scan data (against a real web server). Run the following command in the terminal:

    - **wireshark /tmp/mycap01.pcap;**
      wireshark /home/kali/data/cap00.pcap

2) Two copies of Wireshark will open. Switch between them by clicking on the menu bar at the top of the screen as shown in Fig. 20.

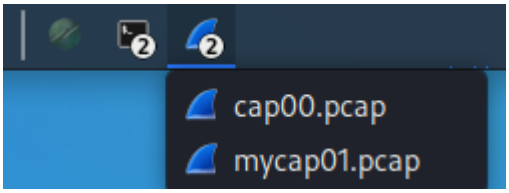

Fig. 20 Wireshark menu items

3) Locate the **first HTTP packet** in the **cap00.pcap** file and then click on **Hypertext Transfer Protocol** in the Packet Layers.

4) Now locate the first HTTP packet in the **mycap01.pcap** file and then click on **Hypertext Transfer Protocol** in the Packet Layers (Fig. 21).

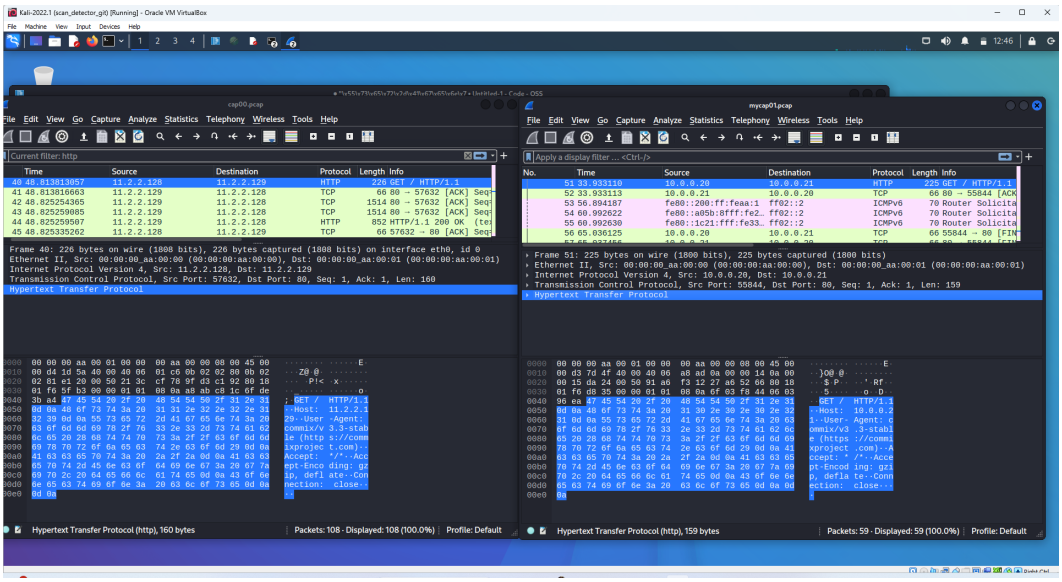

Fig. 21 Side by side captures

5) Notice that in both capture files, there are many similarities, but also a few differences (including the IP addresses 11.2.2.29 vs. 10.0.0.21). To create a good intrusion detection rule, we want to use values that are constant, or the same across multiple runs.

   Since the **User-Agent: commix/v3.3-stable…** field is the same across both (and seems to be very revealing of the Commix tool), let us use this text as our "pattern" for creating an intrusion detection rule.

6) In the Wireshark instance reading **cap00.pcap**, click on the drop-down arrow for the **Hypertext Transfer Protocol** and then click on the **User-Agent** line as shown in Fig. 22.

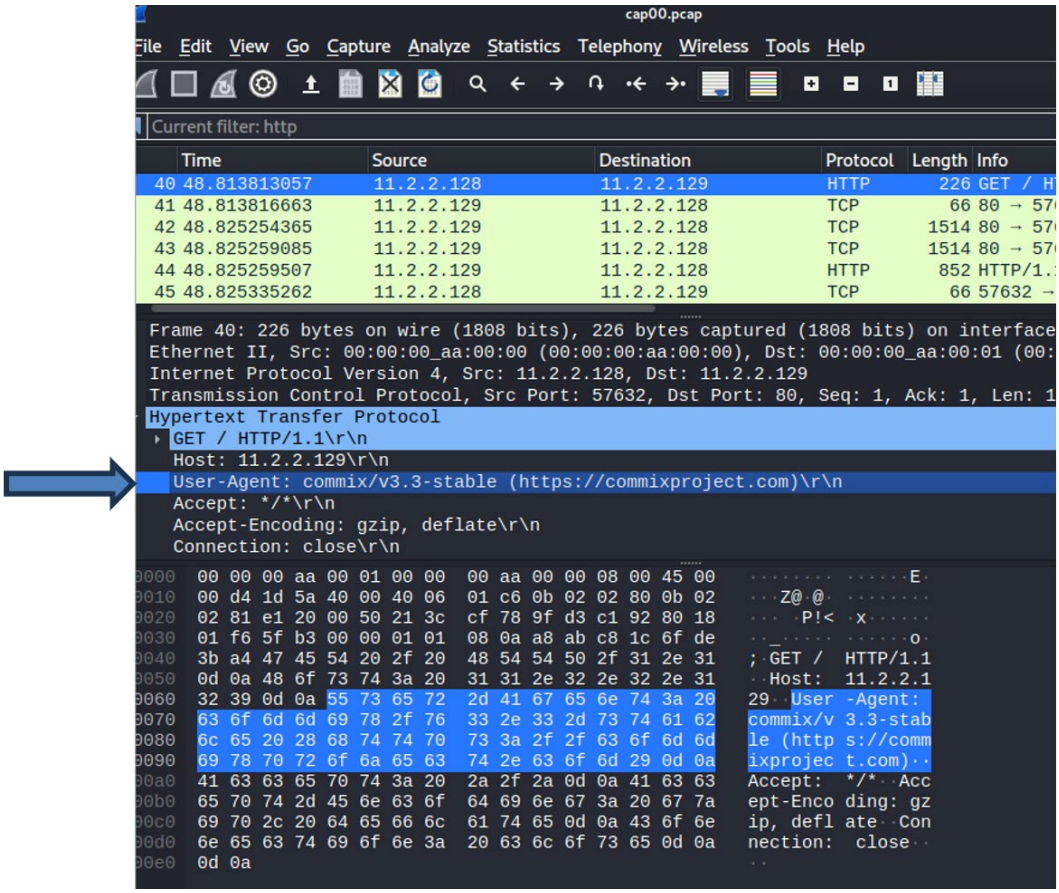

Fig. 22 User-Agent string in Wireshark

7) Right-click on the **User-Agent** line and select **Copy>Value** as shown in Fig. 23.

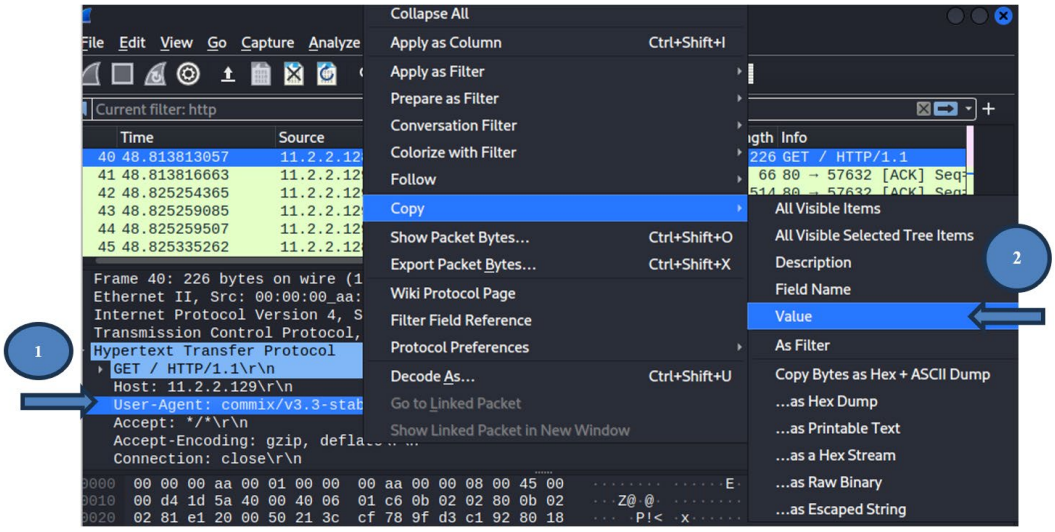

Fig. 23 Copying User-Agent string from Wireshark

8) At this time, you can close (or minimize) all windows on your desktop.

****

Suricata is a widely used, open source, IDS. It has the ability to identify patterns based on rules that adhere to a specific format, on both live traffic as well as captured traffic. Suricata produces alerts when a rule is matched.

A sample rule is as follows:

- alert http any any -> any 80 (msg: "**alert-text**"; content: "**pattern**"; sid: 1000;)

The values above indicate the following:

- **alert**: An alert will be generated when the rules match.
- **http**: Packet must contain HTTP protocol data.
- **any**: Match on any **source address**, **source port**, and **destination address**, respectively.
- **80**: The packet uses port 80 for communication.
- **msg**: The message that will be included with the alert ("alert-text" in this case).
- **content**: The pattern that is checked to match in the packet (the string "pattern" in this case).
- **sid**: A unique identifier associated with the rule.

You can view several rules provided by Emerging Threats at https://rules.emergingthreats.net/open/suricata/rules/.

In the next steps you will use Suricata to identify Commix traffic by editing a prewritten rule template.

****

9) Double-click on the file on the desktop named **commix.rule**. When the file opens, you will see the contents as shown in Fig. 24.

Fig. 24 Sample Suricata rule

10) Paste the text that you copied from the packet (from step 9) inside of the quotes in the **content** field (Fig. 25). Also, change the **msg** field to say "commix_alert" (you can change this to any text).

Fig. 25 Commix Suricata rule

11) Save the file by clicking on **File>Save**.

12) Open a new terminal (Ctrl + Alt + t) and then navigate to the directory with your rule file:

- **cd /home/kali/Desktop/**

The system that you are using has the default Suricata rules (those that are installed after running the Suricata-update command, which contain 35,415 rules at this time).

13) Create a folder to hold results and then run Suricata on your packet capture. First, without the Commix rule you created as follows (the Suricata command may take ~2 min to finish):

- **mkdir default_rules_output**
- **suricata -l ./default_rules_output -r /tmp/mycap01.pcap**
- **mousepad ./default_rules_output/fast.log**

Notice that the **fast.log** file (where alerts are written) is empty.

14) Now, repeat, but with our rule to identify a Commix scan:

- **mkdir custom_commix_rule**
- **suricata -l ./custom_commix_rule -r /tmp/mycap01.pcap -s commix.rule**
- **mousepad ./custom_commix_rule/fast.log**

Now you should see an alert that was generated as a result of the Commix traffic.

Congratulations!

At this time you can close all of your windows. You will start fresh in Section 4.3.

You have just created your own Suricata rule and used it to find Commix traffic in a network packet capture. Recall that even though you used Suricata with its default

rule set, oftentimes there are not any rules for many events; this is because the process can become tedious and difficult, especially if one wants to write a rule that results in **high true-positives** and **low false-positives**.

4.3 Step 3: Automatic Traffic Collection, Analysis, and Rule Generation

In this section, you will learn how to use the GEM tool to automate the process of collecting scanner traffic and creating intrusion detection rules.

The first step in automatic rule creation is choosing and configuring a tool's execution; CORE will be used to create a scenario and collect traffic associated with the tool's scanning traffic.

1) Open another new terminal (Ctrl + Alt + t) and start the CORE-daemon process by executing the following command.

    - **sudo core-daemon**

      *When prompted for a password, enter **kali**.*

    You should see something similar to Fig. 26.

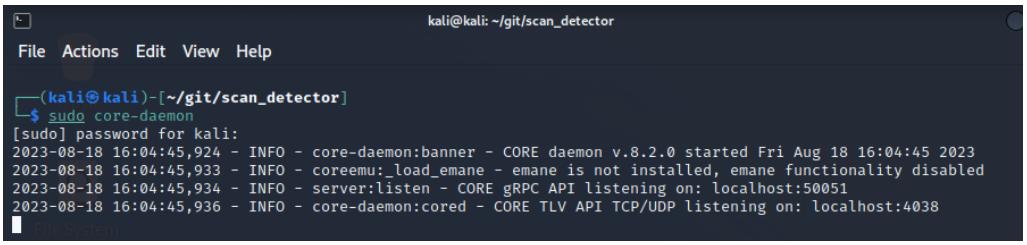

Fig. 26 Start the CORE-daemon

2) Open another new terminal (Ctrl + Alt + t) and open the Visual Studio Code Integrated Development environment by executing the following command:

    - **code**

3) You should be presented with the code window as shown in Fig. 27.

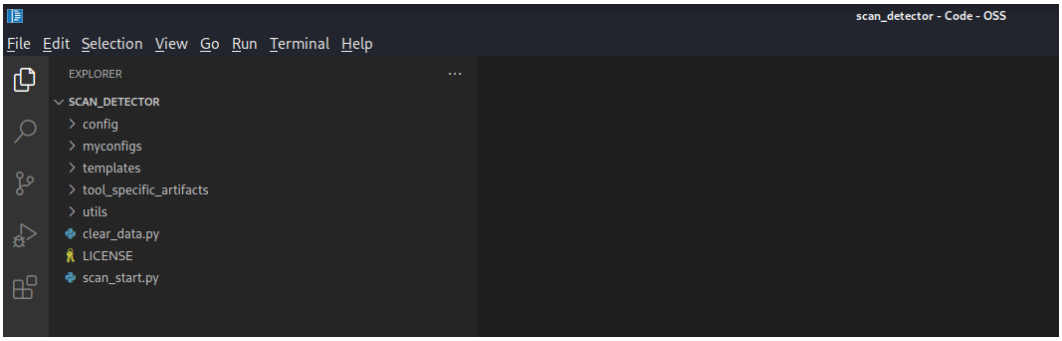

Fig. 27 Open Visual Studio code

If you do NOT see any code files, click on **File->Open Folder** and select **/home/kali/git/scan_detector**.

4) Click on the down arrow next to **myconfigs** and then double-click on the file named **commix_10.0.0.0.cfg**. This will open the first of two configurations for running and collecting data for Commix. You will see the file contents on the right panel as shown in Fig. 28.

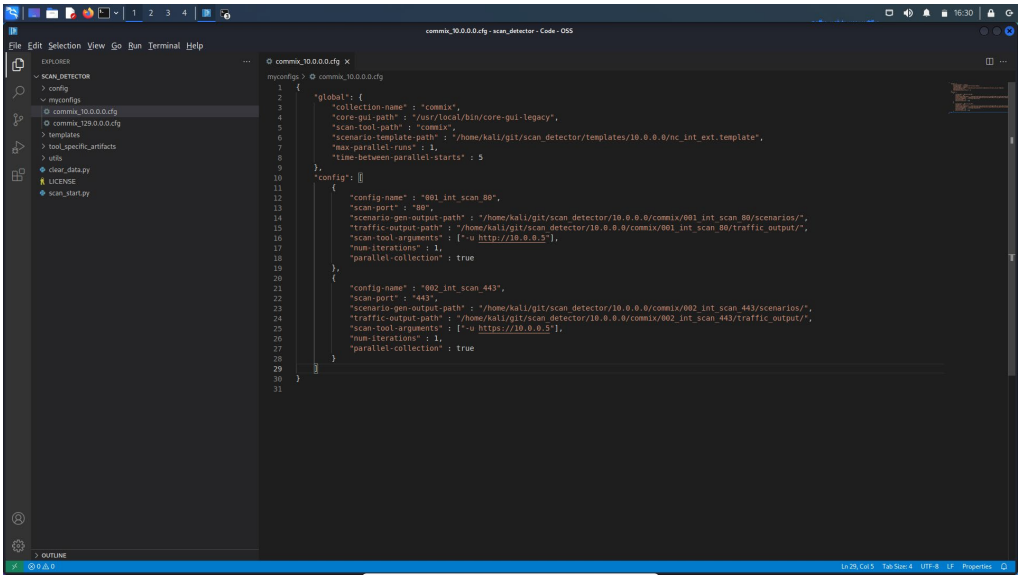

Fig. 28 Sample configuration file

****

Configurations for GEM are highly configurable. The following items are included in the config file:

- **core-gui-path**: Path to the executable for running CORE.
- **scan-tool-path**: Command that will be used to instantiate the scan tool (Commix in this case).

- **scenario-template-path**: Location where the basic CORE scenario file template resides.
- **max-parallel-runs**: Number of scenarios to run simultaneously (work in progress).
- **time-between-parallel-starts**: How long to wait between scenario executions.
- **scan-port**: Remote port to listen for the scan tool connection (can be left out if no listener desired).
- **scenario-gen-output-path**: Location to hold generated CORE scenarios.
- **traffic-output-path**: Location to store network traffic (sender and listener captures).
- **scan-tool-arguments**: Arguments passed to the scan tool when executing (per scan-tool-path).
- **num-iterations**: Number of times to run the scenario (all data for each run is stored).
- **parallel-collection**: Whether to run scenarios in parallel (work in progress).

****

The file that you opened has two configurations that will be executed. Both will scan for web servers, but on different ports (one used for HTTP and the other used for secure HTTP, also known as HTTPS).

5) Based on the configuration file, what IP address will be scanned?

   _____

6) What ports will the victim (the node that will be scanned) listen for connections?

   _____

7) Open the second configuration file by double-clicking the file named **commix_129.0.0.0.cfg**.

Now both configuration files are open and you can switch between them by clicking on the tab at the top of the screen (Fig. 29).

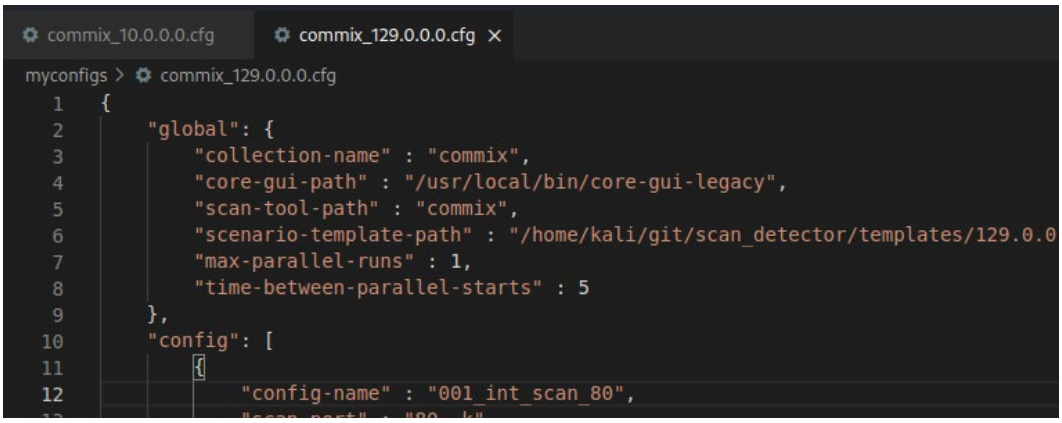

Fig. 29 Customized configuration file

8) Based on the configuration file, what IP address will be scanned?

_____

9) What ports will the victim (the node that will be scanned) listen for connections?

_____

As you can tell, both configuration files are almost identical except for the IP addresses that will be used. Recall from the previous part of this exercise that good intrusion detection rules identify constant patterns that are exhibited by running the tools.

10) Open yet another terminal (Ctrl + Alt + t) and run the configuration file for the 10.x.x.x address by executing the following command:

- **cd /home/kali/git/scan_detector/**
- **python scan_start.py myconfigs/commix_10.0.0.0.cfg**

This will execute a script that will generate a CORE scenario that includes running and collecting data for the tool indicated in the configuration file. It will take roughly 60 s for the Commix configuration but can vary depending on the tool used.

11) Once this finishes, you can look through the files that were generated by opening a file explorer window:

- **open /home/kali/git/scan_detector/10.0.0.0/commix/**

Navigate to the traffic capture files for the **int** scan **port 80** and double-click on the **att.pcap** (i.e., the pcap generated from the node running the Commix scanner; Fig. 30).

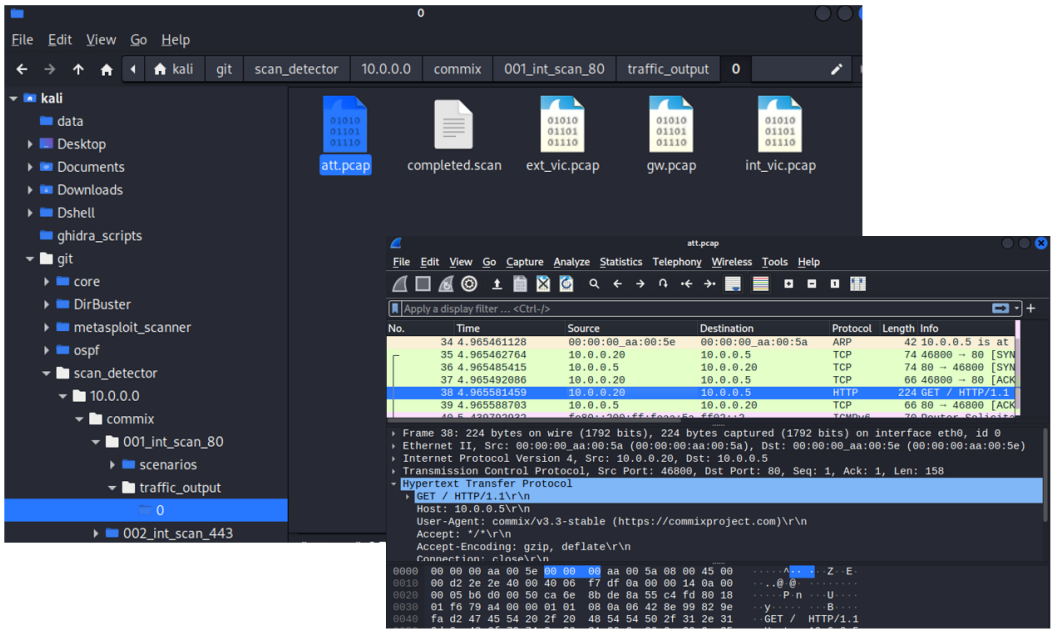

Fig. 30 GEM network capture

At your leisure (optionally), you can also look through the other generated files, including the CORE imn scenarios, node execution scripts, and tool outputs.

12) What is the **Source Address** in the first HTTP packet? (i.e., the address of the node running the Commix scanner).

_____

13) Close Wireshark and the file explorer and then return to your last terminal to run the 129.0.0.0 configuration. Execute the following command:

- **cd /home/kali/git/scan_detector/**
- **python scan_start.py myconfigs/commix_129.0.0.0.cfg**

Again, it will take roughly 60 s for the Commix configuration but can vary depending on the tool used. The output data for this run is located in the following directory: **/home/kali/git/scan_detector/129.0.0.0/commix/**.

14) Now that you have run the multiple configurations of the Commix tool, you have data for the autogeneration of IDS rules. In your terminal window, run the following to process the data:

- **cd /home/kali/git/scan_detector/**
- **python extract_file.py**
- **python analyze_payload.py**

These commands generate several files into a new folder called **processed_data** Now, you will use the file explorer to analyze the newly created files.

15) Open the output folder in file explorer by executing the following command (Fig. 31):

open processed_data/processed_data/commix_001_int_scan_80_traffic_output/0_att/

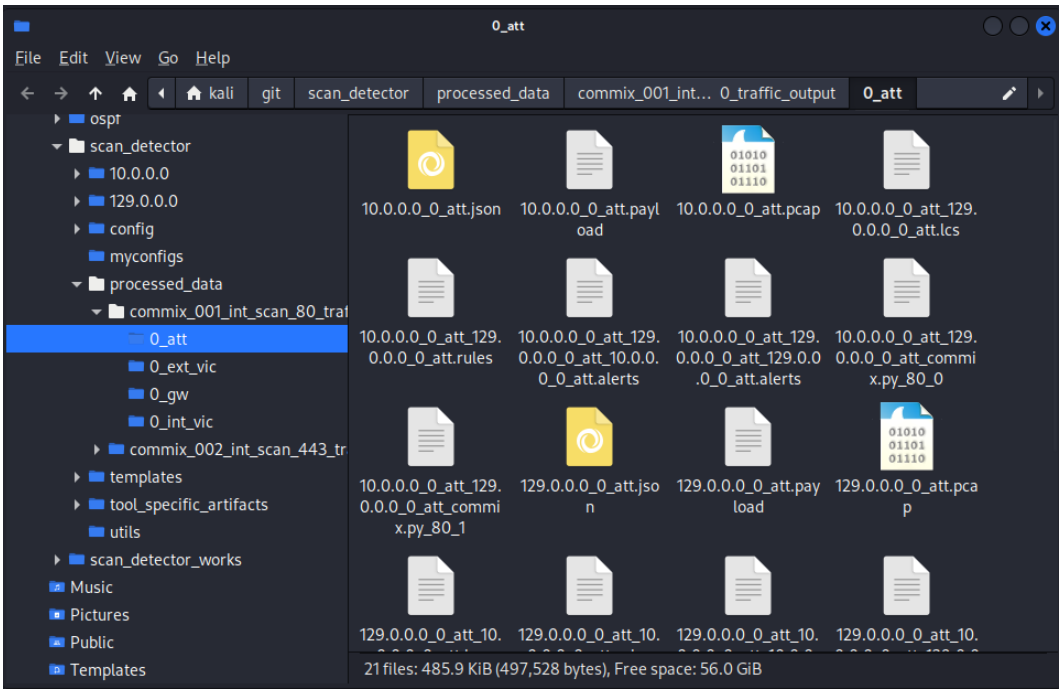

Fig. 31 GEM output files

****

Python's Difflib module has powerful string comparison capabilities. GEM leverages the SequenceMatcher class and functions to identify common strings across the traffic output files—specifically, the longest common substrings. In GEM, these results are stored in files with the "lcs" extension.

****

16) Double-click on the file named **10.0.0.0_0_att_129.0.0.0_0_att.lcs**.

This is a file that contains the common strings in the traffic captures resulting from the many executions of the Commix tool (the file contents are shown in Fig. 32).

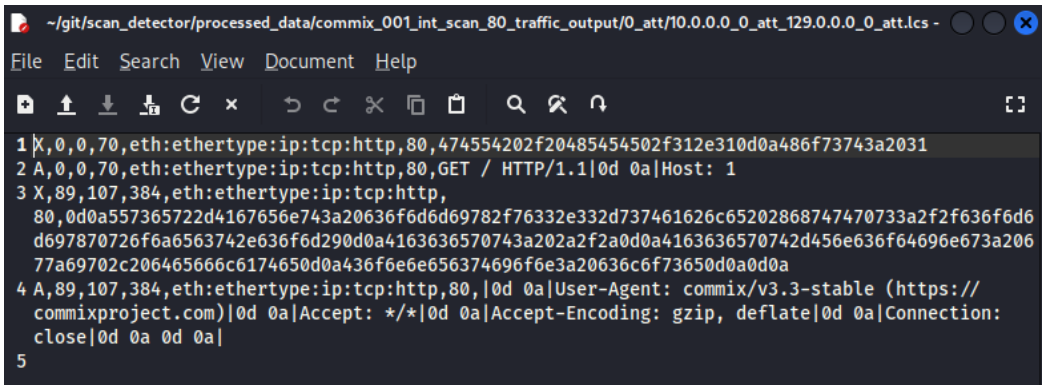

Fig. 32 GEM common substrings

Notice that this file has four rows of data. Lines 1 and 2 represent the same data as lines 3 and 4 using different representations. The first and third lines start with an "X"—meaning that these contain the hexadecimal representation of the data. Lines 2 and 4 contain an "A"—meaning that these contain the ASCII representations of the data.

17) Notice that line 4 contains the pattern you found in the earlier steps of this exercise. Now open the intrusion detection rule that GEM generated by double-clicking on the file named **10.0.0.0_0_att_129.0.0.0_0_att.rules** in your explorer window (Fig. 33).

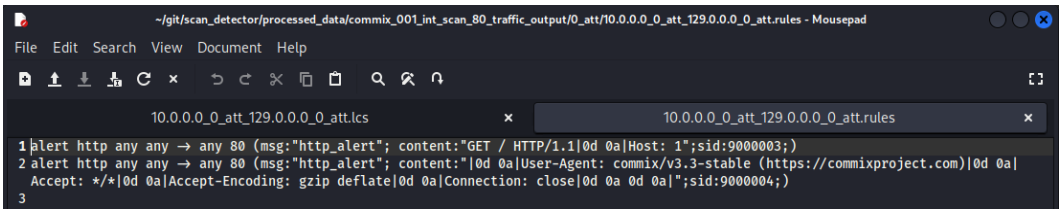

Fig. 33 GEM autogenerated Suricata rules

This file contains the patterns identified in the "lcs" file, but now using the syntax needed by the IDS.

GEM automatically tests the rules generated against the captured traffic using the Suricata IDS.

18) Finally, open the results of running Suricata with your new rules on the captured traffic (Fig. 34). Open the following file:

- **10.0.0.0_0_att_129.0.0.0_0_att_129.0.0.0_0_att.alerts**.

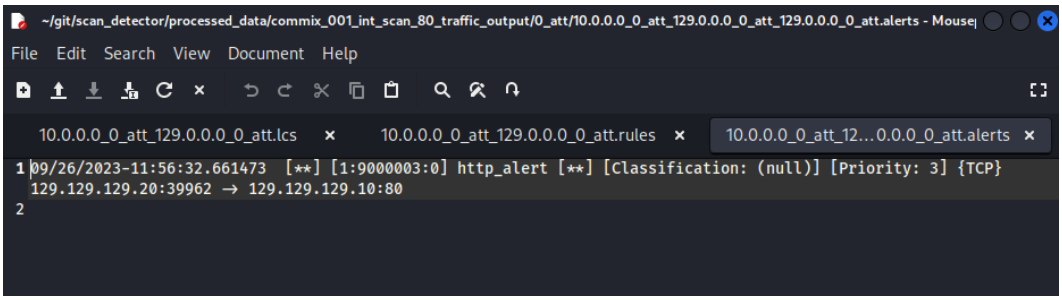

Fig. 34 Suricata alerts for Commix traffic

The file shows the successful detection of the Commix traffic.

Congratulations! You have successfully used GEM to collect traffic for several runs of the Commix tool and then used it to automatically create and test the resulting IDS rules.

Now you can look through some of the other files that GEM generated, especially the .py—these are Python programs that can be used as an alternative to a full-blown IDS. You can also test GEM with some of the other configurations (located in **/home/kali/git/scan_detector/configs**) for other scanning tools.

5. Conclusion

This report provides a learning module focused on network scanning tools and their detection. First, it showed how this process is conducted manually: requiring an analyst to generate a scenario, setup the scanning tool, and then collecting data. The data then needs to be inspected to determine whether any salient patterns exist across many executions of the tool. These patterns can then be used to identify the tool's usage through intrusion detection. The report is meant to be used by analysts with novice-to-advanced cybersecurity and programming backgrounds. Novices will gain firsthand experience, while advanced analysts will have the opportunity to experiment with a state-of-the-art automated scan tool detection system.

6. References

1. Acosta J, Akbar M, Hossain M, Rivas V. Automatic data generation and rule creation for network scanning tools. Proc. of the Future Technologies Conference; 2023.
2. Ahrenholz J, Danilov C, Henderson TR, Kim JH. CORE: a real-time network emulator. IEEE Military Communications Conference (MILCOM); 2008. p. 1–7. IEEE.
3. Suricata. Open Information Security Foundation; c2023 [accessed 2023 Oct]. https://suricata.io/
4. Commix. OffSec Services Limited; c2023 [accessed 2023 Oct]. https://www.kali.org/tools/commix/
5. The JavaScript Object Notation (JSON) Data Interchange Format. IETF Trust; c2014 [accessed 2023 Oct]. https://www.rfc-editor.org/rfc/rfc7159
6. Kali Linux Operating System. OffSec Services Limited; c2023 [accessed 2023 Oct]. https://www.kali.org/
7. Metasploit Framework. Rapid7l c2023 [accessed 2022 July]. https://www.metasploit.com/
8. Masscan. OffSec Services Limited; c2023 [accessed 2023 Oct]. https://www.kali.org/tools/masscan/
9. Nmap. [accessed 2023 Oct]. https://nmap.org/
10. Python. Python Software Foundation; c2023 [accessed 2023 Oct]. https://www.python.org/
11. Difflib. Python Software Foundation; c2023 [accessed 2023 Oct]. https://docs.python.org/3/library/difflib.html
12. VirtualBox. Oracle; c2023 [accessed 2023 Oct]. https://www.virtualbox.org/
13. Visual Studio Code. Microsoft; c2023 [accessed 2023 Oct]. https://code.visualstudio.com/
14. Acosta J, Clarke L, Medina S, Akbar M, Hossain MS, Free-Nelson F. Repeatable experimentation for cybersecurity moving target defense. International Conference on Security and Privacy in Communication Systems; 2021. p. 82–99. Springer, Cham.

List of Symbols, Abbreviations, and Acronyms

CIT	Collaborative Innovation Testbed
CORE	common open research emulator
CyberRIG	Cyber Rapid Innovation Group
GEM	Generate, Examine, Match
GUI	graphical user interface
HTTP	hypertext transport protocol
HTTPS	hypertext transport protocol secure
ICMP	Internet Control Message Protocol
ID	identification
IDS	intrusion detection system
IP	Internet Protocol
JSON	Java Script Object Notation
PC	personal computer
TCP	Transmission Control Protocol
VM	virtual machine
XML	extensible markup language

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CI TECH LIB
1 (PDF)	DEVCOM ARL FCDD RLA ND J ACOSTA