Lab 4

Intrusion Detection - Configuring and Using Snort

Aayush Bulusu

Dr. Bloodworth

JOT 898

IS 6303

December 11, 2024

Table of Contents

Executive Summary ---------- 3

Breakpoint I ---------- 4

Breakpoint II ---------- 8

Breakpoint III ---------- 11

Breakpoint IV ---------- 12

Conclusion ---------- 15

References ---------- 16

Appendix ---------- 17

Executive Summary

Lab 4 is the final lab for this course and deals with Intrusion Detection Systems, also known as IDS, a tool used in cybersecurity that monitors traffic over a network. Intrusion detection devices are known for their ability to thoroughly scrape the network to identify any kind of malicious intrusion into the network or even security policy violations. The benefit of using intrusion detection systems over a network is that it can increase efficiency by automating network threat detection through alerts sent to the system administrator. Intrusion detection systems can also combine with Security Information and Event Management softwares to further combine resources and maximize effectiveness in dealing with potential threats. Moreover, intrusion detection systems also help in ensuring compliance operations. For example, regulations such as the Payment Card Industry Data Security Standard - PCI DSS - require companies that deal with credit card information and therefore need to be compliant to the regulation to implement certain measures such as intrusion detection systems. However, intrusion detection systems do not work to mitigate security threats on their own. They also work in conjunction with intrusion prevention systems - or IPS - which detect and actively work to prevent security threats.

This lab specifically focuses on the SNORT tool, an open-source network intrusion detection tool that works by performing packet logging and real-time traffic analysis, as well as protocol analysis and content searching to detect any sort of intrusion based on its configuration. Using SimSpace, the same platform that the first lab focused on, this lab concentrates on configuring SNORT's settings, capturing and analyzing network traffic, and implementing special rules on a Windows virtual machine as a way to introduce and teach the fundamentals of performing network traffic capture and analysis using an open source intrusion detection tool, as well as emulate some of the real-life processes implemented by cybersecurity professionals in the

industry. This report documents the step-by-step process and analysis on the actions I took throughout the duration of this lab.

Breakpoint I

When running SNORT, it is important to determine the correct network interface SNORT will use. Selecting one interface will result in SNORT running and monitoring traffic on that interface alone. How SNORT is configured determines how well the intrusion detection system works at achieving a certain goal such as how effective it will be at capturing network traffic. Even if all of the other configurations have been properly installed or set, focusing on the incorrect network interface may reduce SNORT's effectiveness. There are many reasons as to why choosing the proper network interface is important for SNORT to properly function.

Identifying Network Traffic

Because SNORT has the special ability to identify all the traffic that passes through a network, choosing the wrong interface will cause it to scan the wrong traffic or gaps in the traffic, causing SNORT to lose its effectiveness at identifying and proactively preventing malicious intrusions into the networks.

Promiscuous Mode

SNORT functions in promiscuous mode, a setting used for instances such as packet sniffing which allows SNORT to read all of the transmitted packets over a network as a way of identifying its behavior and even troubleshoot any issues in the network. The benefit is that it identifies and captures packets that are also not just addressed to the host. Data centers make the best use out of the promiscuous mode due to these locations monitoring heavy traffic on a single interface from multiple systems. Regarding choosing the correct network interface, doing so will

allow SNORT to leverage its promiscuous mode much better and will be able to act as a more effective intrusion detection system over a network.

Physical Interfaces vs Virtual Interfaces

Because this lab makes use of a Windows virtual machine, selecting the right virtual interface gives SNORT its ability to perform traffic capture on packets being transmitted to and from a host system and virtual interface. Not properly identifying the virtual adapter - such as the Net Address Translation or bridged mode - may yield incomplete results since NAT interfaces are restricted to only local traffic on the VM and bridged mode monitors the overall network.

Synergy for Intrusion Prevention Systems Modes

Because SNORT also has the ability to run in in-line mode, it can enable some Intrusion Prevention Systems. In this mode, the network interface that is selected must be compatible for packet injection or bridging, which allows SNORT to actively make changes to the traffic such as modifying or blocking it, or else the intrusion prevention mode will not work nearly as effectively.

Optimizing Performance

Selecting the right interfaces can have varied effects on overall performance. For example, some networks may have higher amounts of high speed traffic, while other networks may be low-performance. Choosing the incorrect interface may cause issues such as dropped packets or packet loss and may hinder SNORT's ability to effectively detect any potential intrusions in the network.

After opening up the command prompt in Windows, I accessed the SNORT directory through a series of commands as shown in Figure 1. When running the command to view the network

interfaces in Windows, I saw three different interfaces with their own internet protocol addresses and device names.

Network Interface Configured for In-Line

When the network interface configuration is set to in-line as an intrusion prevention system rather than the typical intrusion prevention system, there are many changes made to SNORT's functionality that give it new features.

SNORT Traffic Flow

In its intrusion detection mode, SNORT only passively monitors all of the traffic by listening on a chosen interface and does not perform any active functions in the traffic. Typically, the traffic is mirrored to the interface using technologies such as switched port analyzer, or SPAN. In its intrusion prevention, or in-line mode, SNORT switches from a passive role and takes on a more active role. In this configuration, all of the traffic on the network flows through SNORT, allowing it to perform more functions such as modifying, dropping, and inspecting incoming packets based on certain predefined rules.

Network Interface Placement

When set to detection mode, SNORT will be in promiscuous mode which allows it to capture all of the packets transmitted on the network. In this config, SNORT is not placed directly in the path that the traffic flows through. In prevention mode, SNORT requires at least two network interfaces: one for incoming traffic and another for outgoing. These two interfaces are also called ingress and egress respectively and are set up as a bridge, allowing all of the traffic between two network segments to pass through SNORT.

Hardware and Performance

Detection mode configures SNORT to monitor passively so the network interface cards and system are not on the critical path of data flow. Moreover, in this configuration, certain issues such as performance bottlenecks on the monitoring host have little to no effect on the network. In the prevention mode, SNORT is in-line which means that the interfaces and hardware must be able to handle the entire bandwidth of traffic on the network in real-time. Moreover, it is extremely important to have high-performance network interface cards as well as low-latency hardware in order to prevent certain issues arising like delays and dropping packets.

Fail-Safe Mechanisms

In the intrusion detection mode, the failures that SNORT experiences do not disrupt network traffic flow since the system is not directly in the path of the flow and is instead out-of-band. In intrusion prevention mode, the in-line configuration of SNORT may block traffic if SNORT experiences any issues. As a way to mitigate this risk, it is important to implement fail-open interfaces that can bypass SNORT in the event of a failure or issue with the configuration. Moreover, increasing the redundancy by having more network paths for better connection will help maintain traffic flow.

Figures 1b and 1c show the command that I ran and the outputs that followed. The reason I chose the third interface to test the configuration file is that after I asked ChatGPT, it suggested I choose the interface with an internet protocol address similar to the subnet my virtual machine is on in SimSpace. I wanted to capture the network traffic happening on my virtual machine, which is why I chose the third interface with the address 172.16.3.23. After running the command to test the configuration, SNORT returned a message stating that the configuration was successfully

validated. Next, I ran a command to run the pcap using a specific configuration, as shown in Figures 1d and 1e.

Breakpoint II

After running the pcap using a configuration, I proceeded to examine the configuration files and settings of SNORT. As shown in Figure 2a, the notepad that I opened was of the snort.conf file located in the Snort\etc directory. The figure shows the many rules that were included in the text file. Among them are certain rule paths for backdoors, attack responses, app detection, botnets, distributed denial-of-service, exploit kits, and file executable rules. Each of these rules are crucial for the functionality of intrusion detection. For example, the rules for backdoors and botnets are crucial for detecting those two specific types of cyberattacks. Intrusion detection systems rely on a series of rules to properly function. The more advanced the rules are, the more effective the intrusion detection will be. Denial-of-service rules will assist the intrusion detection in better identifying any attempts of denial-of-service attacks happening over a network. Exploit rules can be used by the intrusion detection system as a way of identifying any kind of vulnerabilities that may be taken advantage of or are a weaker target in the eyes of potential hackers. In the text file, there are many more rows of the similar lines of rules. As Figure 2b shows, there are also many rules made on the preprocessor configuration. Preprocessors analyze the traffic before it is matched against the rules. They identify packet streams and are able to perform activities such as reconstructing fragmented packets and inspecting HTTP traffic. Decoder rules monitor and log packet decoding errors which indicate anomalies in the protocols and identify malicious packets. Sensitive data rules identify certain patterns like credit card numbers and other highly personal information, which may prove useful when detecting any

kind of data theft or exfiltration. In the ninth step in the same figure are rules for Shared Objects. Shared Object rules are a series of binary rules that are designed for specific threats. Shared object rules are customizable and are efficient at detecting advanced threats. For example, bad traffic rules are rules for assisting the intrusion detection system in detecting malformed or unexpected traffic patterns. Exploit rules match known exploit traffic targeting specific vulnerabilities in softwares and systems. Miscellaneous rules capture all other miscellaneous abnormalities. These rules provide a new layer of protection and efficiency in SNORT since each rule targets a specific attack or anomaly, which allow system administrators to better focus on these higher priority threats

Intrusion Detection vs Intrusion Prevention Location

By default, SNORT was set to intrusion detection, where its role is more passive than active as it only monitors the traffic in the network and logs certain alerts. In Figure 2c, the snort.conf file contains an include command that is highlighted where it establishes a rule path for other intrusion detection system rules, which are meant to complement the rules already set in place by SNORT in the files. There are many other rules above and below that also complement the intrusion detection capabilities such as those that try to detect phishing campaigns and policy rules which also assists companies in ensuring compliance to regulations and other rules set in place. Figure 2d makes a reference to inline mode. Inline mode is when SNORT is configured to intrusion prevention mode. As mentioned previously, inline mode refers to when the traffic flows directly through SNORT. SNORT takes a more active approach and gains the ability to modify, drop, or reject packets based on certain rules. In the figure, the rule states to drop a packet if its value in the length field - internet protocol, transmission control protocol, or user datagram protocol - is greater than the length of a packet if it is in online mode. According to ChatGPT,

SNORT is in default mode. The reason for this is due to the configurations. If the command “config policy_mode: inline” was present, then SNORT would be in inline/intrusion prevention mode. However, due to the absence of this command, it is still in default mode. Moreover, in Figure 2e are configurations made for DAQ, or data acquisition for inline operations. Among the many examples of the <type> configuration are pcap files such as in the case of the practice.pcap file generated earlier in the lab. Pcap files use the libcap library to capture packets using certain tools such as Wireshark. Dump can be used to write packets to a file. Regarding the “config daq_mode” section, passive refers to intrusion detection mode where the configuration is passive by default. Inline refers to intrusion prevention mode where the network is monitored on a much more active basis.

Proposed Changes

There are a few changes I would make to the configuration files to specify the locations of the analysis rules as well as change SNORT’s role. To change the role from detection to prevention/inline, I would make the following change: config policy_mode: inline, to the snort.conf file to convert it from an intrusion detection to the intrusion prevention configuration. Another change I would make is adding “drop tcp any any -> any 80 (msg:"Blocked malicious HTTP request"; content:"malicious"; sid:1000001; rev:1;),” a modification rule that uses drop or reject rather than alert as a way to take on a much more proactive approach rather than just churning out alert notifications. The third change I would make is by configuring a DAQ, or data acquisition, library for packet processing such as netfilter queuing. The configuration in Figure 2f shows the changes I would make in the notepad, such as configuring the data acquisition library to inline mode to better function as an intrusion prevention. Finally, in order to execute these changes, I would save the file and run the “snort -Q -c /path/to/snort.conf --daq nfq -i eth0”

command where -Q indicates inline mode and the “--daq nfq” section represents the data acquisition module for the intrusion prevention system. To perform all of these operations, I used the win-hunt-19 virtual machine.

Breakpoint III

Part III and Breakpoint III involve capturing and analyzing the network’s traffic. On virtual machine 19 - the computer that I have been using throughout the lab - I made certain changes to the snort.conf text file. In Figures 3a and 3b, I changed the daq_mode setting to passive and deleted the # symbol at the end to remove it as a comment and make it a real configuration. I did the same for Figure 3b where I removed the # symbol to properly configure the output. As shown as Figures 3c and 3d, I ran the command to start SNORT configured for intrusion detection. Next, I opened up the Windows VM 20 as the second computer to perform the next steps and ran SNORT there too. I ran SNORT on that virtual machine, then ran a Telnet command, which did not work. Figure 3g shows my first attempt at running Telnet, with the command coming from asking ChatGPT the best way to approach Telnet network traffic capture. After asking ChatGPT about the issue, it suggested that I try downloading Telnet, however that too returned issues. Because none of these solutions worked, I decided to run Wireshark while I executed the commands on the separate virtual machine. This eventually worked as shown in Figures 3h, 3i, and 3j, I ran all of the required commands which yielded errors. However, despite these errors, Wireshark still recorded this communication. As shown in Figures 3k and 3l, running Wireshark on Ethernet 1 showed a series of communications between virtual machines 172.16.3.23 and 172.16.3.24. Among these protocols are the Address Resolution Protocol, Domain Name System, Transmission Control Protocol, NetBIOS Session Service, and Simple Service

Discovery Protocol. While telnet, FTP, HTTP, SSH, and others did not officially establish connections, Wireshark was still able to intercept the transmissions and record the information while it was running. To exit out of SNORT, I first entered the command Ctrl + C to stop SNORT's execution. To check whether it was still active or not, I executed the command as shown in Figure 2m. Because the command did not return any output, SNORT was successfully stopped.

Breakpoint IV

SNORT Rule Structure

Part IV and Breakpoint IV involve SNORT rules. SNORT rules are a set of rules that dictate certain rules or filters such as the things they identify and what to do when matches are found. SNORT rules equally benefit SNORT's intrusion detection and intrusion prevention modes. According to Crowdstrike, there are two main sections of a basic SNORT rule: the rule header and the rule options or body. Rule headers contain information regarding their function such as the actions they perform upon execution and when a traffic packet matches the rule. Rule headers contain five main components. The first component is the action that is declared in the beginning of the SNORT rule. The second is the IP address of both the source and the destination. The third part is the port numbers which contain information on the source and destination ports. The fourth part is the traffic direction. Traffic direction syntax is dictated by -> and <>, where the former represents a single direction and the latter represents bidirectional traffic flow. The last part of the header is the inspection protocol that can contain information on the internet protocol or IP, internet control message protocol or ICMP, transmission control protocol or TCP, and user datagram protocol or UDP. While SNORT currently supports Layers 3 and 4 of the Open

Systems Interconnection model, SNORT can also be instructed to match traffic rules for application-layer services such as SSL/TLS or secure sockets layer/transport layer security, and HTTP, or hypertext transfer protocol. There are five types of basic SNORT rules: alert, block, drop, pass, and log. Alert rules create alerts when SNORT detects any kind of suspicious activity. Block rules identify and block certain traffic that it deems as harmful from entering the network. Drop rules drop the packets as soon as they match a certain rule. Pass rules automatically pass certain packets through a network without any scrutiny if they meet the rules. Log rules log information or keep a record of the traffic if it matches a rule.

Identified CVEs

CVEs are known as Common Vulnerabilities and Exposures which are a list of popularly known vulnerabilities in cybersecurity. Out of all of the existing common vulnerabilities, I choose EternalBlue and Log4Shell. According to Wikipedia, EternalBlue has its reputation of being developed by the National Security Agency and was used in the infamous WannaCry worldwide ransomware attack in 2017 through a computer worm. Log4Shell is a zero-day vulnerability that was used in the Log4j Java logging utility tool using arbitrary code execution and was reported in 2021, which was said to have affected nearly hundreds of millions of devices. Some of the heavily affected services were Amazon Web Services, Steam, Minecraft Java Edition, iCloud, and Cloudflare, as well as nearly 93% of enterprise cloud environments.

As shown in Figure 4a with Log4Shell, I started with an Alert action as a way to notify or alert in the case of detection in traffic. Next, I wanted to set the protocol to TCP so that it can match the TCP traffic. I set the sources and destination to “any” as a way of making the most use of SNORT and allowing it to monitor all of the network internet protocols and ports for better detection. I selected port 80 not only because it is the default port for HTTP, but also because

port 80 is known to be highly insecure, meaning that it could be an easy way into a network if the port is not scanned properly based on predefined rules. I set the content match to identify any JNDI string. JNDI, or Java Naming and Directory Interface, was one of the two servers - the other being Lightweight Directory Access Protocol - that the vulnerability took advantage of in Log4j during the previously mentioned attack. Specifying these strings could increase the effectiveness of the rule by patching a security flaw that the attack once took advantage of. I also made the rule case insensitive due to the fact that case insensitivity ensures that any capitalized or lowercase string of texts that SNORT combs through will not be an issue for SNORT when it comes to threat identification. Finally, I provided the rule with an identification number and a revision number to make it more identifiable.

In the same figure, the rule for EternalBlue is also visible. Regarding the specific action, protocol, and source/destination, I configured it the same as Log4Shell to increase EternalBlue rule's effectiveness as much as possible by having it scan all ports and addresses for enhanced detection as well as alerts on any issues. Because the EternalBlue attack exploited a vulnerability in Microsoft's Server Message Block or SMB, I configured the port to be 445 since that is where SMB traffic flows through. I set the content match field to search for an SMB header to make it easier for SNORT to identify any potential threats from Log4Shell. I set the offset and depth values to 4 each so that I can limit SNORT's search to look for the SMB header that is typically used in EnternalBlue exploitations. Finally, I gave the EternalBlue rule its own unique identification and revision to make it much easier to find in the rule file next to the Log4Shell rule. It is also a good reference number to use in the case of running any commands that might make use of the identification number.

Process

After entering and saving the new rules in the rules file, I ran the command as shown in Figures 4b and 4c to validate the new configuration. Next, in Figures 4d and 4e, I ran the command that began packet processing to verify that no issues occurred while compiling the new rules. Finally, after this verification, I went back to the rules file, removed the EternalBlue and Log4Shell rules and saved the newly edited document, as shown in Figure 4f to verify that the edits were removed and the document was restored to its original state.

Conclusion

This lab focuses on applying the intrusion detection and prevention tool SNORT to perform intrusion detection. While I found this lab highly difficult and technical, I also found it highly educational and insightful. This lab placed heavy emphasis on the basic commands to start SNORT, analyze SNORT's configuration settings, use Wireshark in conjunction with SNORT, as well as researching and writing different rules for SNORT as a way to identify and patch any Common Vulnerabilities and Exposures. Overall, this lab provided crucial information on the inner workings of intrusion detection and prevention systems and how they go about identifying and mitigating the threats and attacks that happen everyday, and an eye-opening experience into the complexities and infinite potential of these tools that many in the cybersecurity industry use often.

References

Chatgpt. (n.d.-a). https://chatgpt.com/

What is an intrusion detection system (IDS)?. IBM. (2024, August 7). https://www.ibm.com/topics/intrusion-detection-system

Lenaerts-Bergmans, B. (2023, April 24). *Snort explained: Understanding snort rules and use cases*. CrowdStrike. https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/snort-rules/#:~:text=Alert%20rules%3A%20Snort%20generates%20an,as%20the%20alert%20is%20generated.

Wikimedia Foundation. (2024a, November 9). *EternalBlue*. Wikipedia. https://en.wikipedia.org/wiki/EternalBlue

Wikimedia Foundation. (2024a, November 19). *Log4Shell*. Wikipedia. https://en.wikipedia.org/wiki/Log4Shell

Appendix

```
C:\Users\Administrator\Desktop\Snort>cd bin

C:\Users\Administrator\Desktop\Snort\bin>snort -W

          -*> Snort! <*-
  ,,_     Version 2.9.17-WIN32 GRE (Build 199)
 o"  )~   By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
  ''''    Copyright (C) 2014-2020 Cisco and/or its affiliates. All rights reserved.
          Copyright (C) 1998-2013 Sourcefire, Inc., et al.
          Using PCRE version: 8.10 2010-06-25
          Using ZLIB version: 1.2.3

Index   Physical Address        IP Address      Device Name     Description
-----   ----------------        ----------      -----------     -----------
    1   00:00:00:00:00:00       0.0.0.0 \Device\NPF_{1DB0560C-8B99-4D04-8E95-B8A670920723}      MS NDIS 6.0 LoopBack Driver
    2   00:00:00:00:00:00       10.10.0.28      \Device\NPF_{9A7A2A86-BB6C-40B2-B74F-377F03513C60}      VMware vmxnet3 virtual network device
    3   00:00:00:00:00:00       172.16.3.23     \Device\NPF_{B09EB7B3-148C-446C-B745-B2BB82CED37A}      VMware vmxnet3 virtual network device

C:\Users\Administrator\Desktop\Snort\bin>
```

Figure 1a

```
C:\Users\Administrator\Desktop\Snort\bin>snort -i 3 -c c:\Users/Administrator\Desktop\Snort\etc\snort.conf -T
Running in Test mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file "c:\Users/Administrator\Desktop\Snort\etc\snort.conf"
PortVar 'HTTP_PORTS' defined :  [ 80:81 311 383 591 593 901 1220 1414 1741 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144:7145 7510 77
77 7779 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180:8181 8243 8280 8300 8800 8888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443:34444 41080
50002 55555 ]
PortVar 'SHELLCODE_PORTS' defined :  [ 0:79 81:65535 ]
PortVar 'ORACLE_PORTS' defined :  [ 1024:65535 ]
PortVar 'SSH_PORTS' defined :  [ 22 ]
PortVar 'FTP_PORTS' defined :  [ 21 2100 3535 ]
PortVar 'SIP_PORTS' defined :  [ 5060:5061 5600 ]
PortVar 'FILE_DATA_PORTS' defined :  [ 80:81 110 143 311 383 591 593 901 1220 1414 1741 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144
:7145 7510 7777 7779 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180:8181 8243 8280 8300 8800 8888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443
:34444 41080 50002 55555 ]
PortVar 'GTP_PORTS' defined :  [ 2123 2152 3386 ]
Detection:
   Search-Method = AC-Full-Q
    Split Any/Any group = enabled
    Search-Method-Optimizations = enabled
    Maximum pattern length = 20
Tagged Packet Limit: 256
Loading dynamic engine C:\Users\Administrator\Desktop\Snort\lib\snort_dynamicengine\sf_engine.dll... done
Loading all dynamic preprocessor libs from C:\Users\Administrator\Desktop\Snort\lib\snort_dynamicpreprocessor...
  Loading dynamic preprocessor library C:\Users\Administrator\Desktop\Snort\lib\snort_dynamicpreprocessor\sf_dce2.dll... done
  Loading dynamic preprocessor library C:\Users\Administrator\Desktop\Snort\lib\snort_dynamicpreprocessor\sf_dnp3.dll... done
  Loading dynamic preprocessor library C:\Users\Administrator\Desktop\Snort\lib\snort_dynamicpreprocessor\sf_dns.dll... done
```

Figure 1b

```
Acquiring network traffic from "\Device\NPF_{B09EB7B3-148C-446C-B745-B2BB82CED37A}".

        --== Initialization Complete ==--

          -*> Snort! <*-
  ,,_     Version 2.9.17-WIN32 GRE (Build 199)
 o"  )~   By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
  ''''    Copyright (C) 2014-2020 Cisco and/or its affiliates. All rights reserved.
          Copyright (C) 1998-2013 Sourcefire, Inc., et al.
          Using PCRE version: 8.10 2010-06-25
          Using ZLIB version: 1.2.3

          Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 3.1  <Build 1>
          Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
          Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
          Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
          Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
          Preprocessor Object: SF_SDF  Version 1.1  <Build 1>
          Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
          Preprocessor Object: SF_POP  Version 1.0  <Build 1>
          Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
          Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
          Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
          Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
          Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
          Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
          Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>

Snort successfully validated the configuration!
Snort exiting

C:\Users\Administrator\Desktop\Snort\bin>
```

Figure 1c

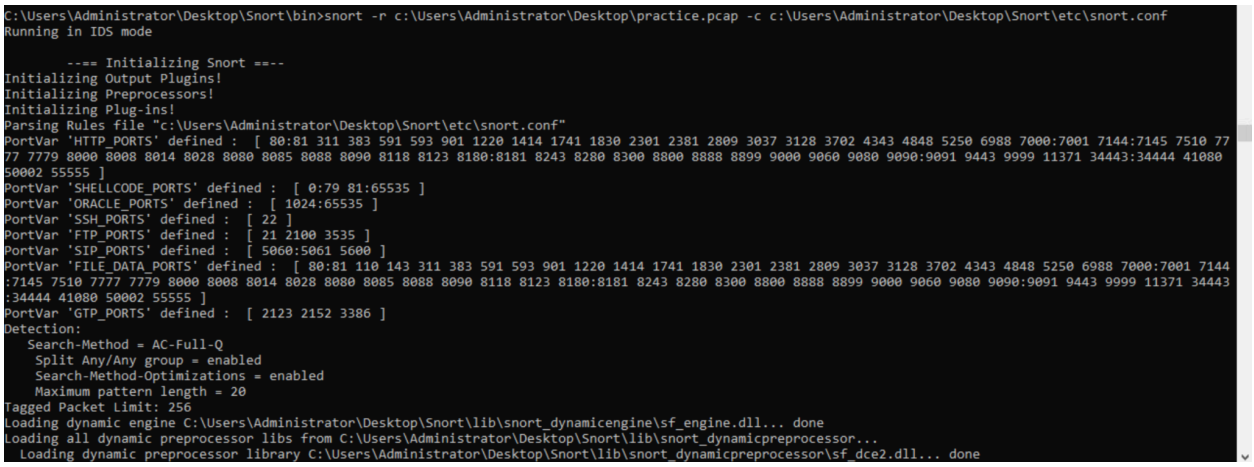

Figure 1d

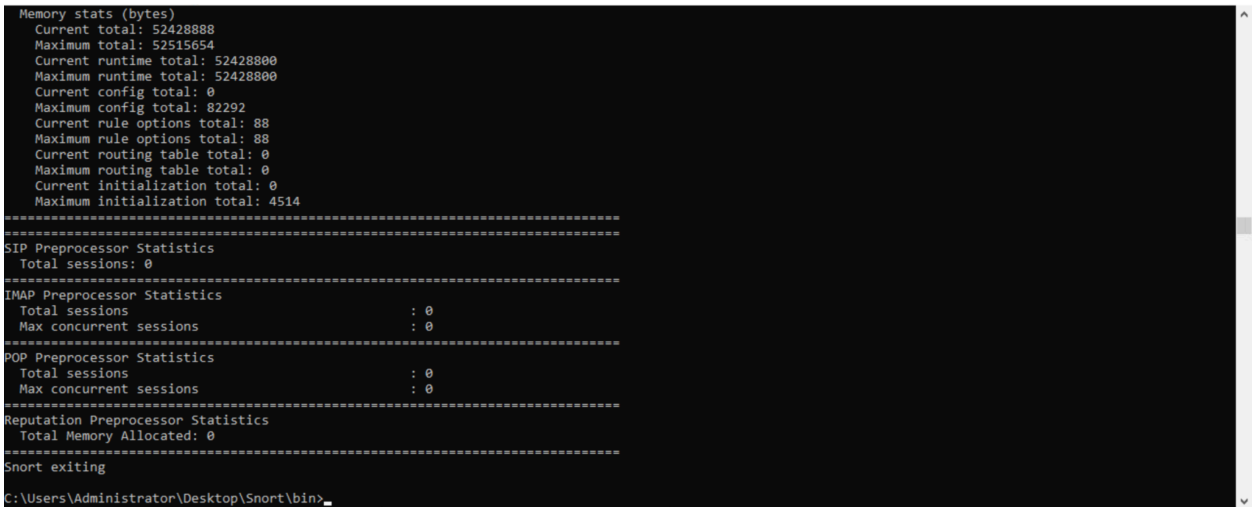

Figure 1e

```
include $RULE_PATH\app-detect.rules
include $RULE_PATH\attack-responses.rules
include $RULE_PATH\backdoor.rules
include $RULE_PATH\bad-traffic.rules
include $RULE_PATH\blacklist.rules
include $RULE_PATH\botnet-cnc.rules
include $RULE_PATH\browser-chrome.rules
include $RULE_PATH\browser-firefox.rules
include $RULE_PATH\browser-ie.rules
include $RULE_PATH\browser-other.rules
include $RULE_PATH\browser-plugins.rules
include $RULE_PATH\browser-webkit.rules
include $RULE_PATH\chat.rules
include $RULE_PATH\content-replace.rules
include $RULE_PATH\ddos.rules
include $RULE_PATH\dns.rules
include $RULE_PATH\dos.rules
include $RULE_PATH\experimental.rules
include $RULE_PATH\exploit-kit.rules
include $RULE_PATH\exploit.rules
include $RULE_PATH\file-executable.rules
include $RULE_PATH\file-flash.rules
```

Figure 2a

```
##################################################
# Step #8: Customize your preprocessor and decoder alerts
# For more information, see README.decoder_preproc_rules
##################################################

# decoder and preprocessor event rules
include $PREPROC_RULE_PATH\preprocessor.rules
include $PREPROC_RULE_PATH\decoder.rules
include $PREPROC_RULE_PATH\sensitive-data.rules

##################################################
# Step #9: Customize your Shared Object Snort Rules
# For more information, see http://vrt-blog.snort.org/2009/01/using-vrt-certified-shared-obje
##################################################

# dynamic library rules
# include $SO_RULE_PATH/bad-traffic.rules
# include $SO_RULE_PATH/chat.rules
# include $SO_RULE_PATH/dos.rules
# include $SO_RULE_PATH/exploit.rules
# include $SO_RULE_PATH/icmp.rules
# include $SO_RULE_PATH/imap.rules
# include $SO_RULE_PATH/misc.rules
# include $SO_RULE_PATH/multimedia.rules
# include $SO_RULE_PATH/netbios.rules
# include $SO_RULE_PATH/nntp.rules
```

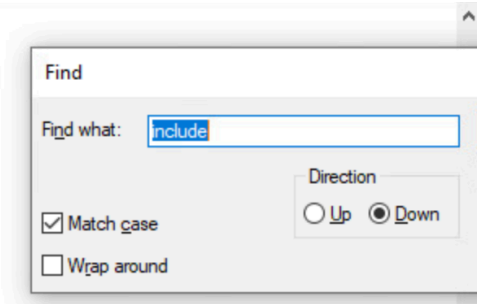

Figure 2b

```
include $RULE_PATH\nntp.rules
include $RULE_PATH\oracle.rules
#include $RULE_PATH\os-linux.rules
#include $RULE_PATH\os-other.rules
#include $RULE_PATH\os-solaris.rules
#include $RULE_PATH\os-windows.rules
include $RULE_PATH\other-ids.rules
include $RULE_PATH\p2p.rules
include $RULE_PATH\phishing-spam.rules
include $RULE_PATH\policy-multimedia.rules
#include $RULE_PATH\policy-other.rules
include $RULE_PATH\policy.rules
include $RULE_PATH\policy-social.rules
include $RULE_PATH\policy-spam.rules
```

Figure 2c

```
# Same as above, but drop packet if in Inline mode (requires enable_decode_oversized_alerts)
# config enable_decode_oversized_drops
```

Figure 2d

```
# Configure DAQ related options for inline operation. For more information, see README.daq
#
# config daq: <type>
# config daq_dir: <dir>
# config daq_mode: <mode>
# config daq_var: <var>
#
# <type> ::= pcap | afpacket | dump | nfq | ipq | ipfw
# <mode> ::= read-file | passive | inline
# <var> ::= arbitrary <name>=<value passed to DAQ
# <dir> ::= path as to where to look for DAQ module so's
```

Figure 2e

```
config daq: nfq
config daq_mode: inline
config daq_var: queue=0
```

Figure 2f

```
# config daq: <type>
# config daq_dir: <dir>
config daq_mode: passive
# config daq_var: <var>
```

Figure 3a

```
output alert_unified2: filename snort.alert, limit 128, nostamp
```

Figure 3b

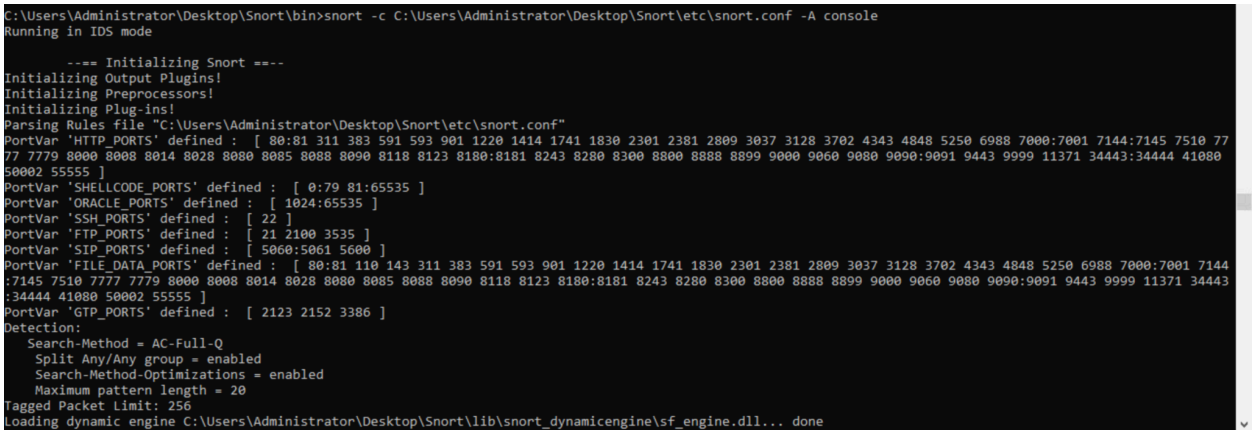

Figure 3c

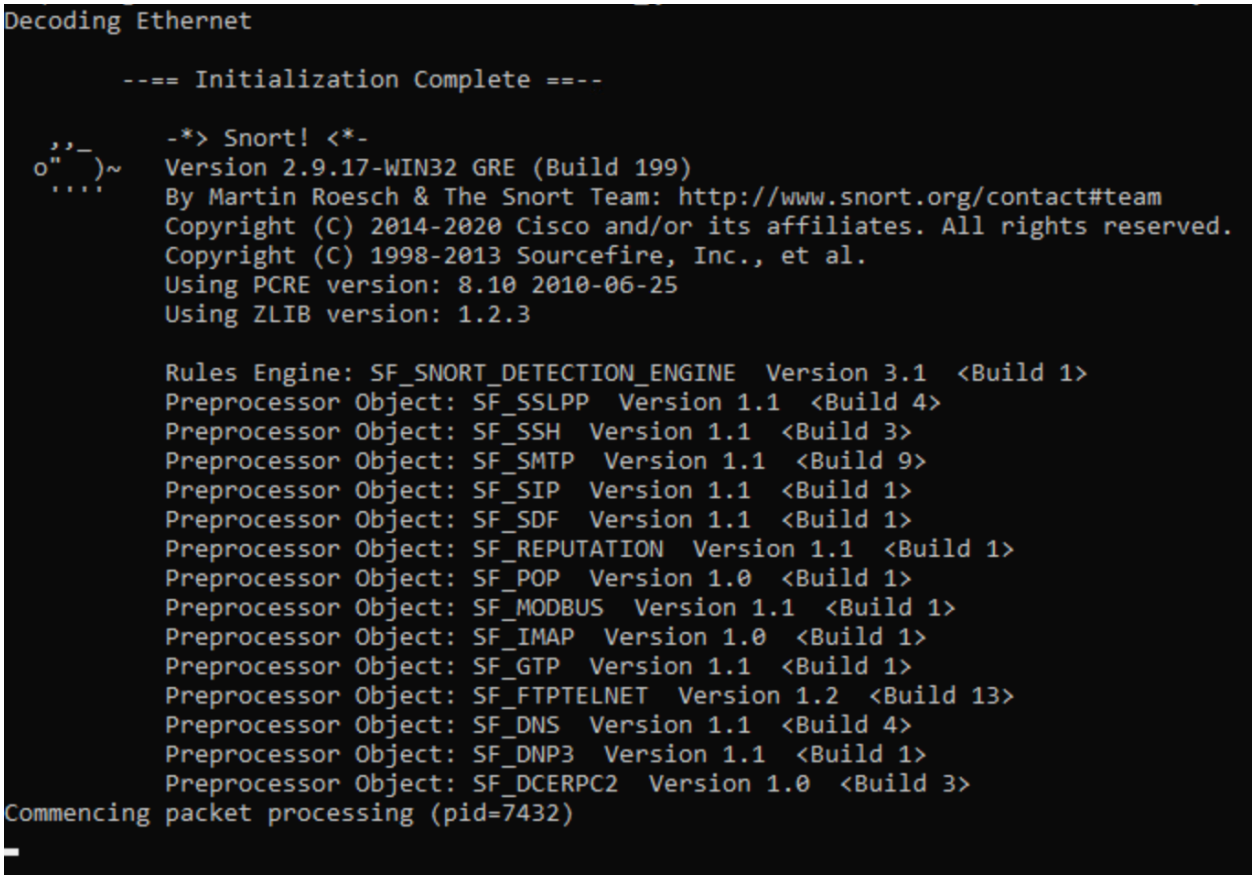

Figure 3d

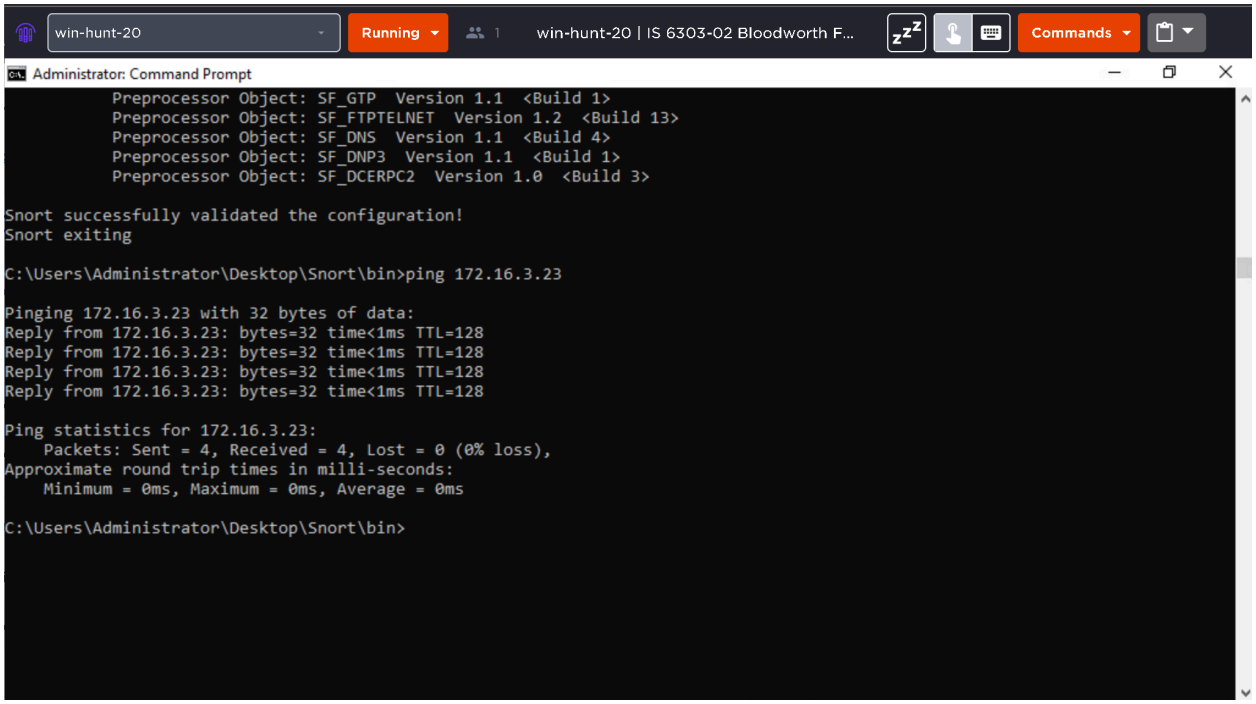

Figure 3f

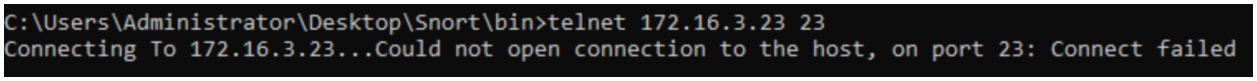

Figure 3g

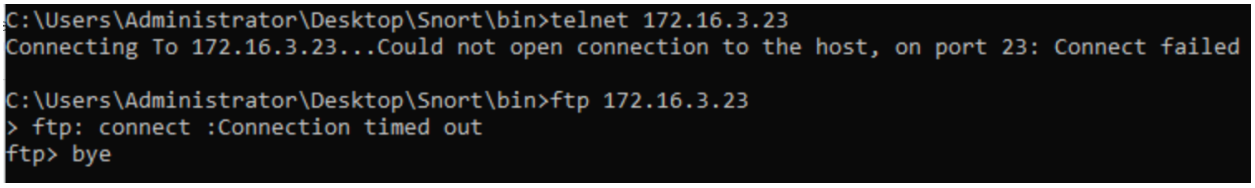

Figure 3h

```
C:\Users\Administrator\Desktop\Snort\bin>ssh 172.16.3.23
ssh: connect to host 172.16.3.23 port 22: Connection timed out

C:\Users\Administrator\Desktop\Snort\bin>ping 172.16.3.23

Pinging 172.16.3.23 with 32 bytes of data:
Reply from 172.16.3.23: bytes=32 time<1ms TTL=128
Reply from 172.16.3.23: bytes=32 time<1ms TTL=128
Reply from 172.16.3.23: bytes=32 time<1ms TTL=128
Reply from 172.16.3.23: bytes=32 time<1ms TTL=128

Ping statistics for 172.16.3.23:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Figure 3i

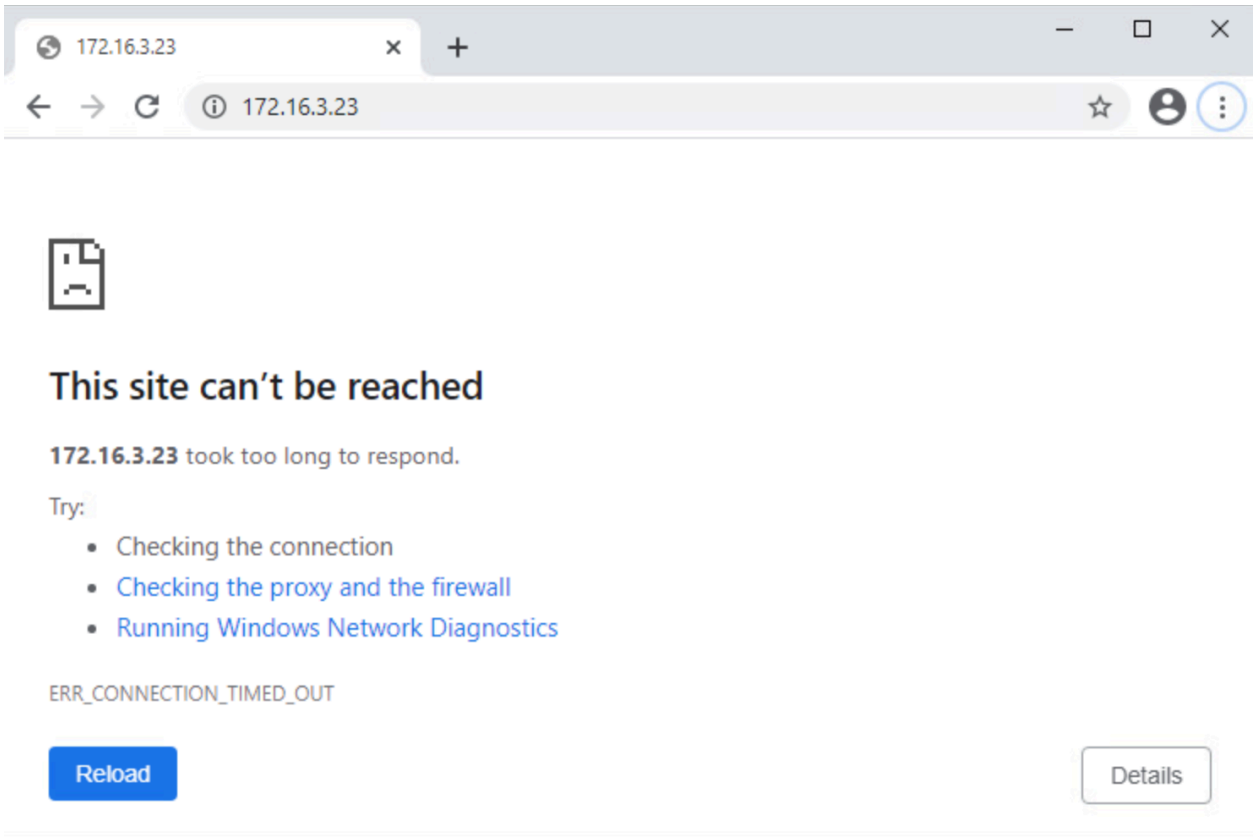

Figure 3j

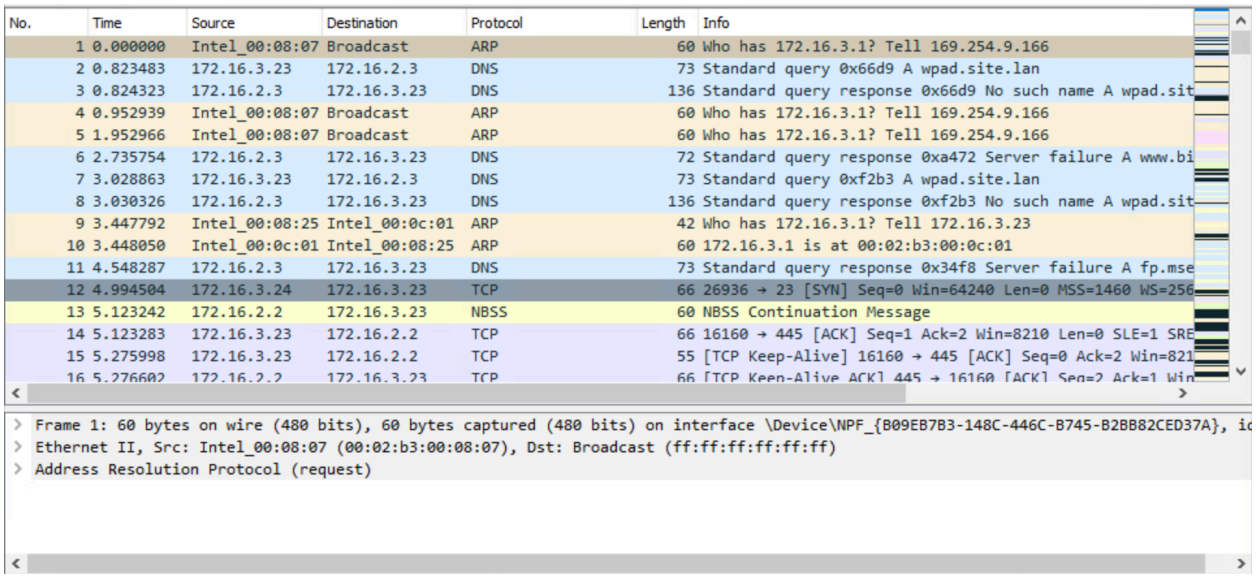

Figure 3k

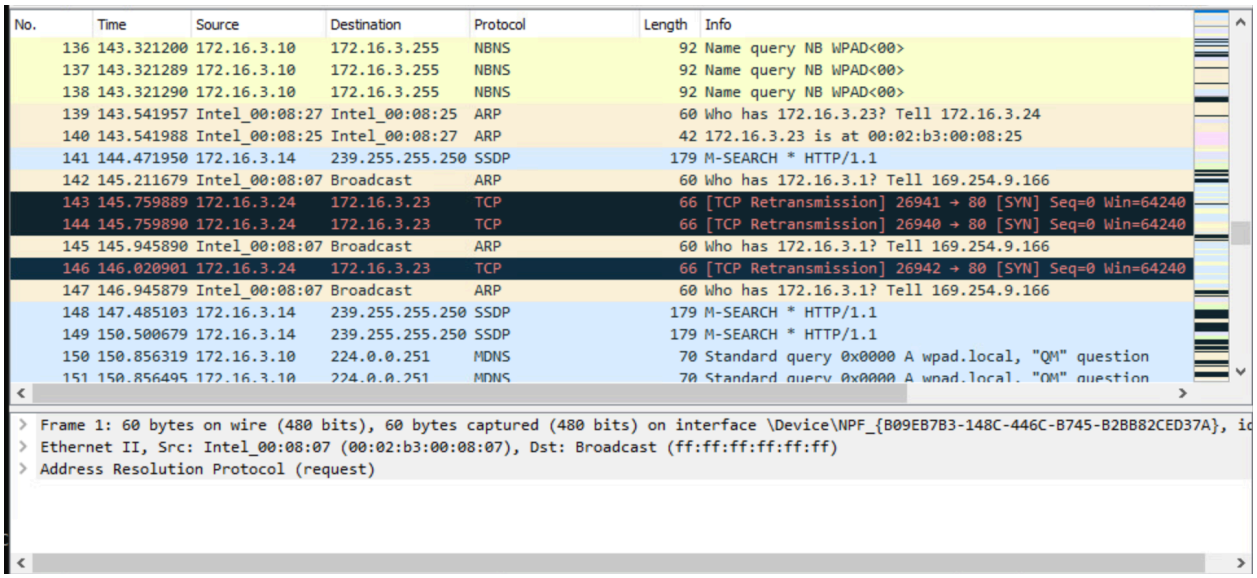

Figure 3l

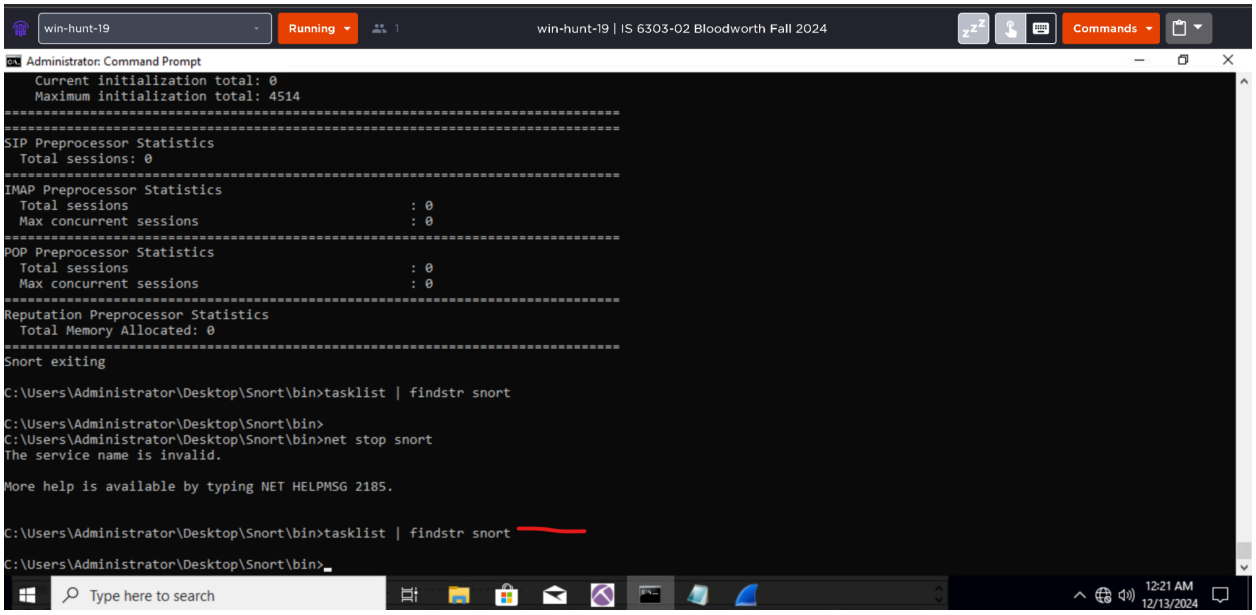

Figure 3m

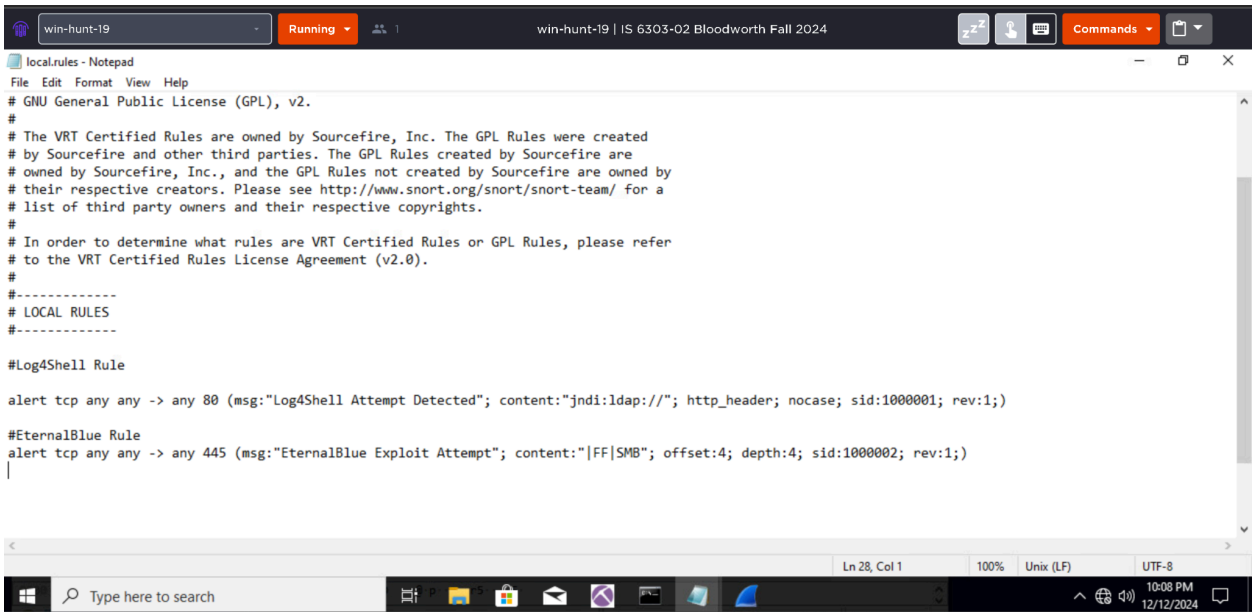

Figure 4a

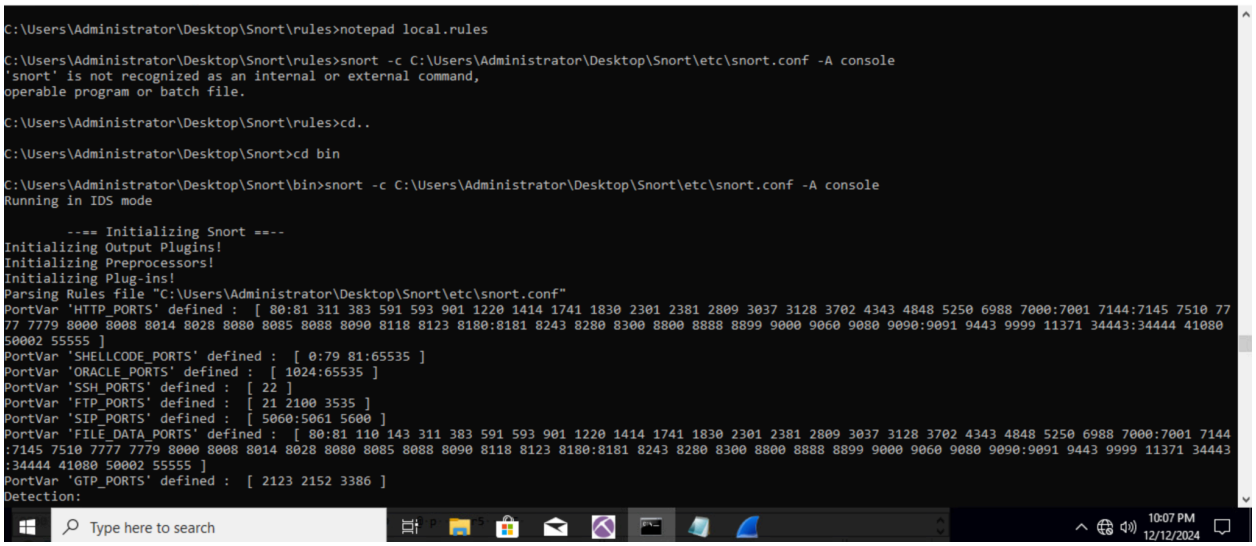

Figure 4b

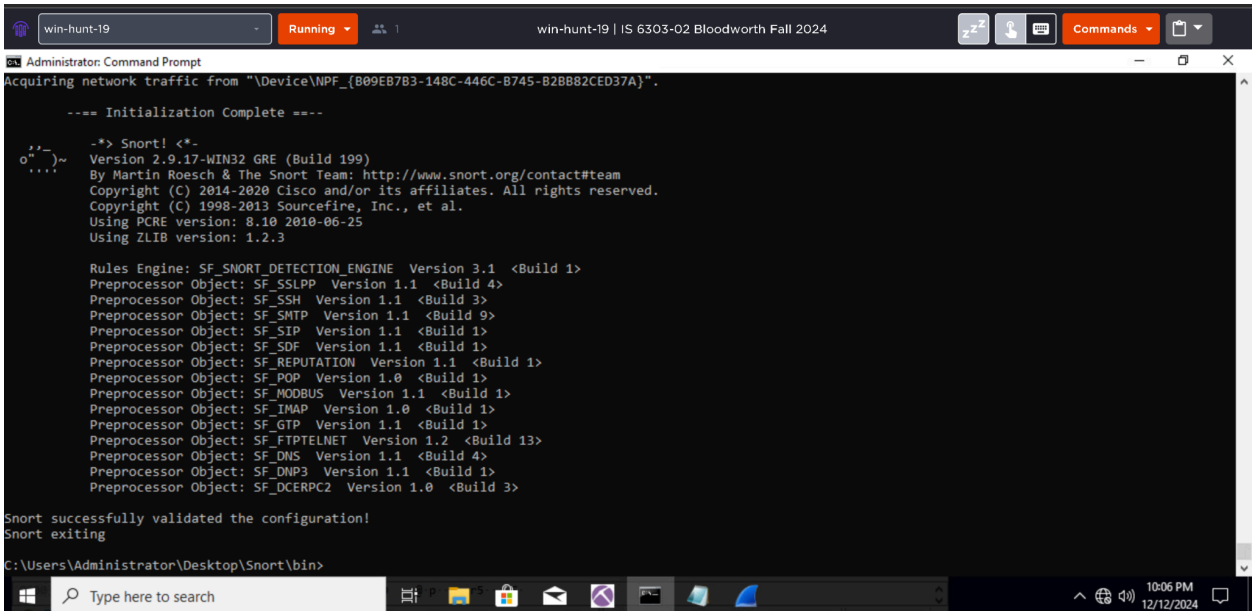

Figure 4c

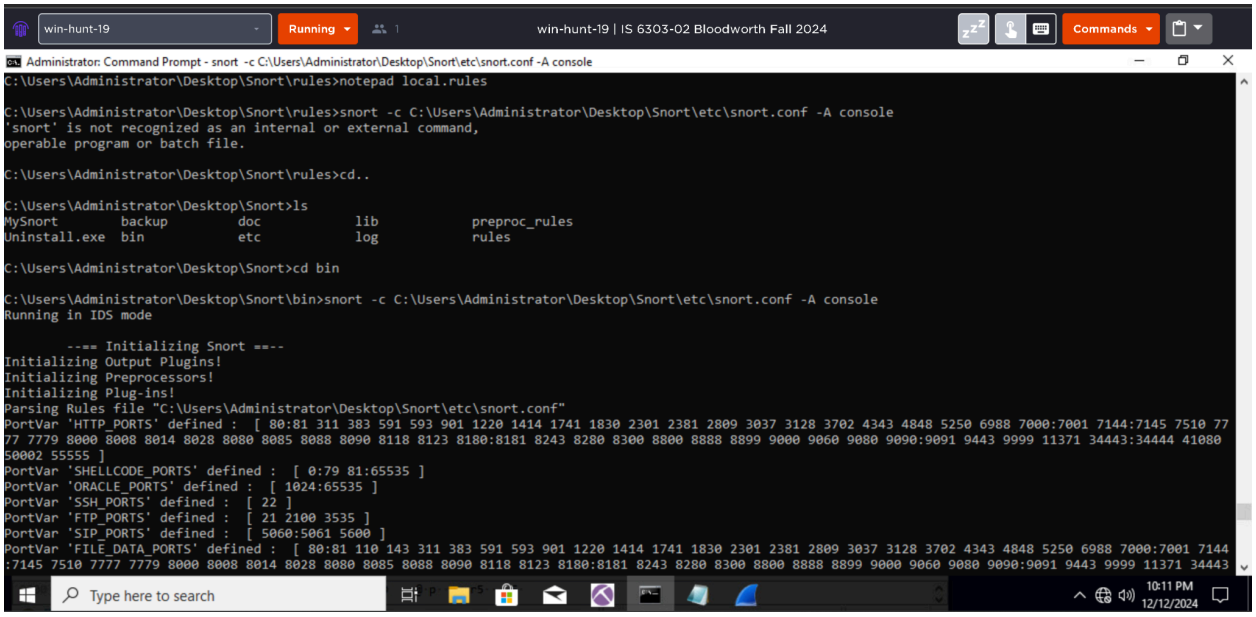

Figure 4d

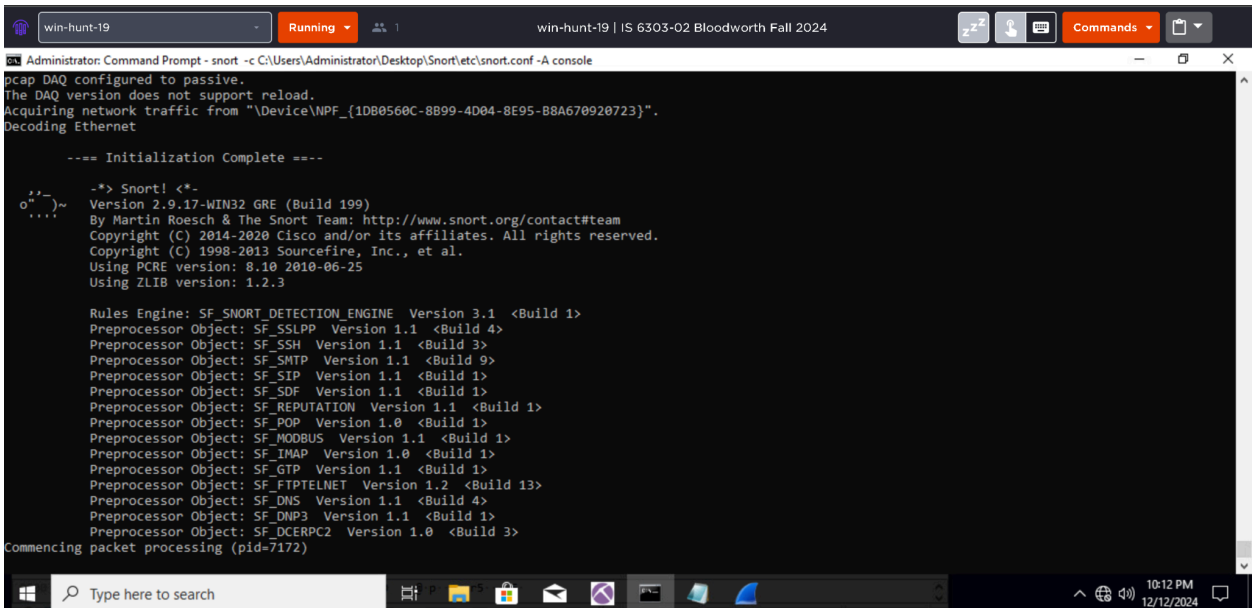

Figure 4e

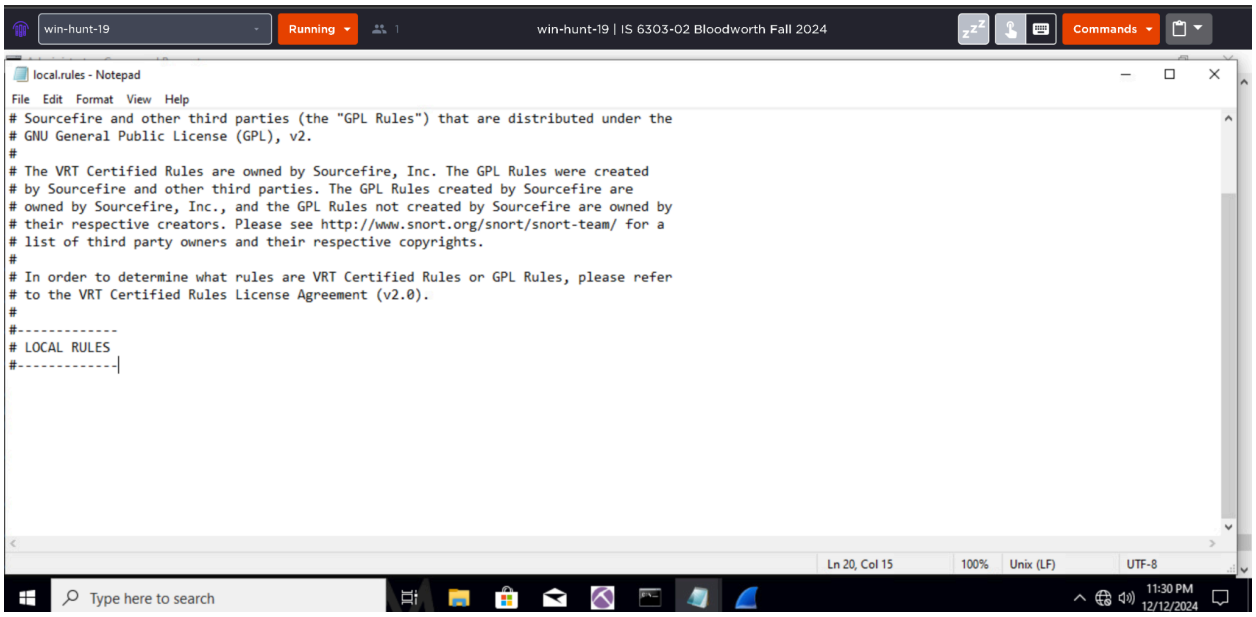
win-hunt-19
Running
win-hunt-19 | IS 6303-02 Bloodworth Fall 2024
Commands
local.rules - Notepad
File Edit Format View Help
Sourcefire and other third parties (the "GPL Rules") that are distributed under the
GNU General Public License (GPL), v2.
#
The VRT Certified Rules are owned by Sourcefire, Inc. The GPL Rules were created
by Sourcefire and other third parties. The GPL Rules created by Sourcefire are
owned by Sourcefire, Inc., and the GPL Rules not created by Sourcefire are owned by
their respective creators. Please see http://www.snort.org/snort/snort-team/ for a
list of third party owners and their respective copyrights.
#
In order to determine what rules are VRT Certified Rules or GPL Rules, please refer
to the VRT Certified Rules License Agreement (v2.0).
#
#-------------
LOCAL RULES
#-------------
Ln 20, Col 15
100%
Unix (LF)
UTF-8
Type here to search
11:30 PM
12/12/2024

Figure 4f