

Lab 5-Network Monitoring

Dawn M Inman

CSC-432 Computer and Network Security

Professor Nick Merante

April 17, 2020

### Abstract

This lab sets up tcpdump on the router and goes through several small data exercises like comparing traffic on the public and private interfaces and isolating SSH and HTTP traffic.

Argus is also set up on the router through quite a few steps: get and install, downloading source code packages, compiling, building the argus-clients package, configure the service by adding to the /etc/argus.conf file, create argus user and groups, set permissions, start the service and verify that it is running and view the network flow. Two different argus clients were used for filtering, rfilteraddr and rasort.

*Keywords:* CentOS 7, Kali Xfce, Proxmox, QEMU, PuTTY, tcpdump, argus, HTTP, SSH, ratop, rfilteraddr, rasort.

### Lab 5-Network Monitoring

Network traffic can be complicated to look at and understand. Two tools that can help with that are tcpdump and argus. These tools are both widely used and can be used easily from the command line. Both tools are standards for cybersecurity programs.

The open source program, tcpdump is a command-line tool that captures and analyzes network traffic that flows through the system, in this case the router. It has multiple options and filters so it can be used for most anything. It can be run on systems with a GUI and without, so it is ideal for those servers that do not have one. Data is often collected to be analyzed later. It can also be scheduled by using other tools along with it, which shows its flexibility. There are many online sources for information about tcpdump but a good one is supported by Red Hat, called opensource.com, which gives a great introduction to tcpdump as well as simple installation instructions for multiple platforms. (Gerardi, 2018)

Argus is an open source tool that is a network flow system that has been adapted for cybersecurity. It addresses several network flow data issues like privacy, performance, applicability and utility. It processes packets into data that can be easily read very quickly. The main system, called argus, is what generates the data and the argus-clients are a collection of data processing programs. Argus was created in the early 1980's at Georgia Tech and is still a critical component of cybersecurity. A new website was just built for Argus and its' clients, released in October 2019. (argus, 2020)

This lab uses the virtual router on the student network, a CentOS 7 router. CentOS 7 is so named because it stands for Community ENTERprise Operating System. It is based on the Linux kernel, free and has been available since 2004. Red Hat Enterprise Linux is the origination of CentOS 7 so it is a compatible option when requiring Linux software. It is very

popular with almost 30% of Linux web servers using it in 2011 and has been one of the most popular in hosting history. (CentOS Blog, 2020)

Kali Xfce is a newer Kali release. It is on the same line of Kali environments that have been created for Penetration Testing. There is a new feature called “Kali Undercover” which can make the display of Kali look like Windows 10. This can happen quickly so it is a type of stealth feature meant for blending in when in public areas. (Abrams, 2019) Other new features include KaliNetHunter KeX for Android which can install a full Kali desktop via Android, upgrading the kernel, Git powered documentation and adding PowerShell. (Elwood, 2019)

Proxmox is also being used by the systems and is used extensively in this lab. Proxmox VE hypervisor is based on GNU/Linux (Debian) and is open source. It has a central web-based management that does not require more installation. (Cheng, 2014) Version 5.4 is built specifically on Debian 9.8 with a “specially modified Linux Kernel 4.15”. (Proxmox, 2019) Proxmox is capable of two types of virtualization: OpenVZ and KVM. OpenVZ needs a patched Linux kernel so Linux guests are the only operating system type that can be created. In OpenVZ, the guests are called containers because they share the same architecture and kernel as the host operating system. (Cheng, 2014) KVM (Kernel-based Virtual Machine) is a modified Linux kernel built with the KVM module so that it can give hardware-assisted virtualization. Virtualization is performed by a software-based emulator (QEMU) which simulates the virtualized environment while KVM only exposes the /dev/kvm interface. (Cheng, 2014) “This converts Linux into a Type 1 (bare-metal) hypervisor.” (What is KVM?, 2020) Then QEMU or the software-based emulator will create the virtual machines on top of KVM. (What is KVM? 2020) Proxmox VE is relatively simple to start working with but can be very in depth as Simon

M.C. Cheng has authored a book called Proxmox High Availability which goes into more detail when setting up a high availability virtual cluster. (Cheng, 2014)

PuTTY is used several times during the lab for double checking past router command. PuTTY is an SSH client for Windows, Mac and Linux. It has a terminal window for access to the server used in this lab, the GNU/Linux server named router. (How to use PuTTY on Windows, 2020) SSH is a software package and means Secure Shell. It secures system administration and file transfers even though the networks are insecure. Tatu Ylonen is the inventor of SSH and OpenSSH which is an open source SSH program is based off of his free versions. (SSH(Secure Shell), 2020)

### **Objective**

This lab's purpose is to get the router VM ready to watch traffic on the network. This is accomplished through the installation of tcpdump, which shows the packets that move through the router, and argus, which shows the network flow in real time as it moves through the router.

The computer that is being used is a 2011 HP Pavillion dv7, i7 quad core processor and 16GB RAM with Windows 10 Pro operating system. Google Chrome is the internet browser being used for connecting to Proxmox. KaliVM is on Proxmox. The KaliVM then runs Fire Fox internet browser, PuTTY as the SSH connection. The router is a CentOS 7 system with tcpdump and argus as the router network tools being experimented with during the lab.

### **Results and Analysis<sup>1</sup>**

#### **Tcpdump**

For this lab, tcpdump needed to be installed on the router. Using the instructions tcpdump was installed on the router with

Yum install tcpdump

To put the same program onto the KaliVM, it needs to be typed into the command line differently.

```
apt-get update && apt-get install tcpdump
```

The man page (manual) is found at

```
man tcpdump
```

To listen to a certain interface, the option needs to be listed as -i. Two different interfaces were desired, the internal router interface, eth1 and the external or gateway, eth0. The commands are below with a portion of what was returned during the session:

```
tcpdump -i eth0
```

```

13:47:16.585499 IP a23-10-96-34.deploy.static.akamaitechnologies.com.https > router.41576: Flags [P.], seq 2584:2906, ack 1878,
win 256, options [nop,nop,TS val 2274910287 ecr 2527854911], length 727
13:47:16.585603 IP router.41576 > a23-10-96-34.deploy.static.akamaitechnologies.com.https: Flags [.], ack 1857, win 497, options
[nop,nop,TS val 2527855014 ecr 2274910287], length 0
13:47:16.585614 IP router.41576 > a23-10-96-34.deploy.static.akamaitechnologies.com.https: Flags [.], ack 2584, win 492, options
[nop,nop,TS val 2527855014 ecr 2274910287], length 0
13:47:16.585619 IP router.41576 > a23-10-96-34.deploy.static.akamaitechnologies.com.https: Flags [.], ack 2906, win 490, options
[nop,nop,TS val 2527855014 ecr 2274910287], length 0
13:47:18.290532 STP 802.1d, Config, Flags [none], bridge-id 800a.c8:00:84:00:89:80.801a, length 42
13:47:20.293415 STP 802.1d, Config, Flags [none], bridge-id 800a.c8:00:84:00:89:80.801a, length 42
13:47:20.570103 IP router.42576 > a96-6-27-12.deploy.static.akamaitechnologies.com.https: Flags [P.], seq 610:641, ack 153, win
501, options [nop,nop,TS val 4218305356 ecr 2374323531], length 31
13:47:20.570129 IP router.42576 > a96-6-27-12.deploy.static.akamaitechnologies.com.https: Flags [F.], seq 641, ack 153, win 501,
options [nop,nop,TS val 4218305356 ecr 2374323531], length 0
13:47:20.586173 IP a96-6-27-12.deploy.static.akamaitechnologies.com.https > router.42576: Flags [.], ack 641, win 235, options [
nop,nop,TS val 2374328545 ecr 4218305356], length 0
13:47:20.586209 IP a96-6-27-12.deploy.static.akamaitechnologies.com.https > router.42576: Flags [P.], seq 153:184, ack 642, win
235, options [nop,nop,TS val 2374328545 ecr 4218305356], length 31
13:47:20.586235 IP a96-6-27-12.deploy.static.akamaitechnologies.com.https > router.42576: Flags [F.], seq 184, ack 642, win 235,
options [nop,nop,TS val 2374328545 ecr 4218305356], length 0
13:47:20.586355 IP router.42576 > a96-6-27-12.deploy.static.akamaitechnologies.com.https: Flags [R], seq 500790323, win 0, lengt
h 0
13:47:20.586367 IP router.42576 > a96-6-27-12.deploy.static.akamaitechnologies.com.https: Flags [R], seq 500790323, win 0, lengt
h 0
13:47:20.882176 IP router.55178 > 65.52.54.98.https: Flags [.], ack 22096, win 524, options [nop,nop,TS val 931052040 ecr 151861
1161], length 0
13:47:20.920558 IP 65.52.54.98.https > router.55178: Flags [.], ack 2006, win 514, options [nop,nop,TS val 151862123 ecr 9310426
151], length 0
13:47:22.296256 STP 802.1d, Config, Flags [none], bridge-id 800a.c8:00:84:00:89:80.801a, length 42
^C
563 packets captured
675 packets received by filter
112 packets dropped by kernel
[roo@router etc]$

```

Figure 1 eth0 tcpdump

```
tcpdump -i eth1
```

```

ack 1896, win 256, options [nop,nop,TS val 2258910407 ecr 1147775519], length 1020
13:53:13.736662 IP 192.168.11.10.46222 > a23-217-96-106.deploy.static.akamaitechnologies.com.https: Flags [I], ack 1857, win 497
, options [nop,nop,TS val 1147775607 ecr 2258910407], length 0
13:53:13.736684 IP 192.168.11.10.46222 > a23-217-96-106.deploy.static.akamaitechnologies.com.https: Flags [I], ack 2877, win 490
, options [nop,nop,TS val 1147775607 ecr 2258910407], length 0
13:53:17.998389 IP 192.168.11.10.42624 > a96-6-27-12.deploy.static.akamaitechnologies.com.https: Flags [P], seq 610:641, ack 15
3, win 501, options [nop,nop,TS val 4218662785 ecr 2374681018], length 31
13:53:17.998469 IP 192.168.11.10.42624 > a96-6-27-12.deploy.static.akamaitechnologies.com.https: Flags [F], seq 641, ack 153, w
in 501, options [nop,nop,TS val 4218662785 ecr 2374681018], length 0
13:53:18.015088 IP a96-6-27-12.deploy.static.akamaitechnologies.com.https > 192.168.11.10.42624: Flags [I], ack 641, win 235, op
tions [nop,nop,TS val 2374685979 ecr 4218662785], length 0
13:53:18.015126 IP a96-6-27-12.deploy.static.akamaitechnologies.com.https > 192.168.11.10.42624: Flags [P], seq 153:184, ack 64
1, win 235, options [nop,nop,TS val 2374685979 ecr 4218662785], length 31
13:53:18.015132 IP a96-6-27-12.deploy.static.akamaitechnologies.com.https > 192.168.11.10.42624: Flags [F], seq 184, ack 642, w
in 235, options [nop,nop,TS val 2374685979 ecr 4218662785], length 0
13:53:18.015325 IP 192.168.11.10.42624 > a96-6-27-12.deploy.static.akamaitechnologies.com.https: Flags [R], seq 3584708775, win
0, length 0
13:53:18.015346 IP 192.168.11.10.42624 > a96-6-27-12.deploy.static.akamaitechnologies.com.https: Flags [R], seq 3584708775, win
0, length 0
13:53:18.033167 IP 192.168.11.10.55224 > 65.52.54.98.https: Flags [I], ack 22096, win 524, options [nop,nop,TS val 931409192 ecr
1520510381], length 0
13:53:18.063753 IP 65.52.54.98.https > 192.168.11.10.55224: Flags [I], ack 2022, win 514, options [nop,nop,TS val 152052042 ecr
9313991811], length 0
13:53:23.045277 IP 192.168.11.10.46214 > a23-217-96-106.deploy.static.akamaitechnologies.com.https: Flags [I], ack 6007, win 501
, options [nop,nop,TS val 1147784915 ecr 2258909714], length 0
13:53:23.085560 IP a23-217-96-106.deploy.static.akamaitechnologies.com.https > 192.168.11.10.46214: Flags [I], ack 2769, win 269
, options [nop,nop,TS val 2258919757 ecr 1147774913], length 0
13:53:23.505278 IP 192.168.11.10.46220 > a23-217-96-106.deploy.static.akamaitechnologies.com.https: Flags [I], ack 3668, win 501
, options [nop,nop,TS val 1147785375 ecr 2258910126], length 0
13:53:23.505330 IP 192.168.11.10.46218 > a23-217-96-106.deploy.static.akamaitechnologies.com.https: Flags [I], ack 6868, win 501
, options [nop,nop,TS val 1147785375 ecr 2258910026], length 0
13:53:23.545696 IP a23-217-96-106.deploy.static.akamaitechnologies.com.https > 192.168.11.10.46220: Flags [I], ack 1195, win 244
, options [nop,nop,TS val 2258920217 ecr 1147775325], length 0
13:53:23.545730 IP a23-217-96-106.deploy.static.akamaitechnologies.com.https > 192.168.11.10.46218: Flags [I], ack 1206, win 244
, options [nop,nop,TS val 2258920217 ecr 1147775225], length 0
^C
464 packets captured
640 packets received by filter
176 packets dropped by kernel
root@router etc1#

```

Figure 2 eth1 tcpdump

There are two differences between the interfaces to point out. When the KaliVM searches the internet, eth1(internal) shows the IP address of the machine but eth0 (external) shows the address as coming from the router instead. This keeps the IP address of the KaliVM private from online sources that could use the information maliciously. Another difference is how much traffic is coming into eth0 because it is the external router. The packets are requests are not always answered by the router as the packets don't apply to it, but it still creates a constant stream of incoming traffic.

To change the setting so that only SSH traffic is seen, the command line needs to be:

```
tcpdump port 22
```

```

[root@router etc]# tcpdump port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:19:13.710615 IP router.44087 > 10.42.0.1.ssh: Flags [S], seq 599531684, win 64240, options [mss 1460,sackOK,TS val 2770335030,
  ecr 0,nop,wscale 7], length 0
13:19:13.712214 IP 10.42.0.1.ssh > router.44087: Flags [S.], seq 1820228793, ack 599531685, win 4128, options [mss 1460], length
  0
13:19:13.712332 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 1, win 64240, length 0
13:19:13.714115 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 1:20, ack 1, win 4128, length 19
13:19:13.714226 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 20, win 64221, length 0
13:19:13.809933 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 1:29, ack 20, win 64221, length 28
13:19:13.809952 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 29:1197, ack 20, win 64221, length 1168
13:19:13.811285 IP 10.42.0.1.ssh > router.44087: Flags [.], ack 1197, win 2932, length 0
13:19:13.811506 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 20:324, ack 1197, win 2932, length 304
13:19:13.811576 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 324, win 63917, length 0
13:19:13.811743 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 1197:1221, ack 324, win 63917, length 24
13:19:13.812907 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 324:604, ack 1221, win 2908, length 280
13:19:13.813018 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 604, win 63917, length 0
13:19:13.843079 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 1221:1493, ack 604, win 63917, length 272
13:19:13.905084 IP 10.42.0.1.ssh > router.44087: Flags [.], ack 1493, win 4128, length 0
13:19:14.020464 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 604:1180, ack 1493, win 4128, length 576
13:19:14.020519 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 1180:1196, ack 1493, win 4128, length 16
13:19:14.020689 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 1180, win 63917, length 0
13:19:14.020719 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 1196, win 63917, length 0
13:19:18.705747 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 1493:1597, ack 1196, win 63917, length 104
13:19:18.707472 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 1196:1248, ack 1597, win 4024, length 52
13:19:18.707667 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 1248, win 63917, length 0
13:19:30.694802 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 1597:1701, ack 1248, win 63917, length 104
13:19:30.696368 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 1248:1332, ack 1701, win 3920, length 84
13:19:30.696670 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 1332, win 63917, length 0
13:19:30.696840 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 1701:1837, ack 1332, win 63917, length 136
13:19:30.698336 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 1332:1400, ack 1837, win 3784, length 68
13:19:30.698534 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 1400, win 63917, length 0
13:19:40.473789 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 1837:2137, ack 1400, win 63917, length 300
13:19:40.674177 IP 10.42.0.1.ssh > router.44087: Flags [.], ack 2137, win 3484, length 0
13:19:41.474223 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 1400:1484, ack 2137, win 3484, length 84
13:19:41.474556 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 1484, win 63917, length 0
13:19:41.474767 IP router.44087 > 10.42.0.1.ssh: Flags [P.], seq 2137:2273, ack 1484, win 63917, length 136
13:19:41.476370 IP 10.42.0.1.ssh > router.44087: Flags [P.], seq 1484:1552, ack 2273, win 3348, length 68
13:19:41.476540 IP router.44087 > 10.42.0.1.ssh: Flags [.], ack 1552, win 63917, length 0
^C
95 packets captured
95 packets received by filter
0 packets dropped by kernel
[root@router etc]#

```

Figure 3 eth0 ssh traffic

This shows the traffic for login information from the KaliVM to darknet.

To see the http traffic the command line needs to read what is on port 80.

```
tcpdump port 80
```



```

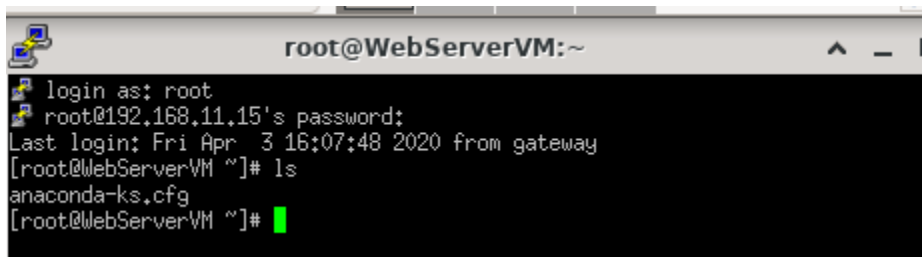
ecr 1573586321, length 0
13:28:40.793114 IP router.33436 > lga25s62-in-f3.1e100.net.http: Flags [I], ack 1405, win 501, options [nop,nop,TS val 414688596
7 ecr 3435396863], length 0
13:28:40.816336 IP lga25s62-in-f3.1e100.net.http > router.33460: Flags [I], ack 435, win 240, options [nop,nop,TS val 157368872
ecr 4146845272], length 0
13:28:40.816375 IP lga25s62-in-f3.1e100.net.http > router.33436: Flags [I], ack 869, win 244, options [nop,nop,TS val 3435407103
ecr 4146845243], length 0
13:28:41.049027 IP router.50020 > 72.21.91.29.http: Flags [I], ack 800, win 501, options [nop,nop,TS val 371320430 ecr 239020942
9], length 0
13:28:41.049056 IP router.50056 > 151.139.128.14.http: Flags [I], ack 1835, win 501, options [nop,nop,TS val 2872033577 ecr 4288
335090], length 0
13:28:41.064683 IP 72.21.91.29.http > router.50020: Flags [I], ack 434, win 285, options [nop,nop,TS val 2390219669 ecr 37127967
3], length 0
13:28:41.064714 IP 151.139.128.14.http > router.50056: Flags [I], ack 862, win 243, options [nop,nop,TS val 4288345330 ecr 28719
92062], length 0
13:28:44.080989 IP router.34032 > cloudproxy10036.sucuri.net.http: Flags [I], ack 4574, win 501, options [nop,nop,TS val 2800961
716 ecr 3051895387], length 0
13:28:44.080917 IP router.34120 > cloudproxy10036.sucuri.net.http: Flags [I], ack 2287, win 501, options [nop,nop,TS val 2800961
716 ecr 3051895387], length 0
13:28:44.915014 IP cloudproxy10036.sucuri.net.http > router.34032: Flags [I], ack 846, win 61, options [nop,nop,TS val 305190562
ecr 2800941416], length 0
13:28:44.915047 IP cloudproxy10036.sucuri.net.http > router.34120: Flags [I], ack 423, win 59, options [nop,nop,TS val 305190562
ecr 2800941442], length 0
13:28:45.400929 IP router.39190 > a23-199-63-49.deploy.static.akamaitechnologies.com.http: Flags [I], ack 914, win 501, options
[nop,nop,TS val 798660370 ecr 2826679236], length 0
13:28:45.400969 IP router.39192 > a23-199-63-49.deploy.static.akamaitechnologies.com.http: Flags [I], ack 914, win 501, options
[nop,nop,TS val 798660370 ecr 2826679235], length 0
13:28:45.419382 IP a23-199-63-49.deploy.static.akamaitechnologies.com.http > router.39192: Flags [I], ack 444, win 235, options
[nop,nop,TS val 2826689475 ecr 798609333], length 0
13:28:45.419483 IP a23-199-63-49.deploy.static.akamaitechnologies.com.http > router.39190: Flags [I], ack 444, win 235, options
[nop,nop,TS val 2826689476 ecr 798609333], length 0
13:28:46.168932 IP router.54362 > server-13-225-69-194.ewr53.r.cloudfront.net.http: Flags [I], ack 2013, win 501, options [nop,n
op,TS val 438368284 ecr 1042254044], length 0
13:28:46.184521 IP server-13-225-69-194.ewr53.r.cloudfront.net.http > router.54362: Flags [I], ack 881, win 122, options [nop,no
p,TS val 1042255068 ecr 438347999], length 0
13:28:46.425008 IP router.49938 > 72.21.91.29.http: Flags [I], ack 1215, win 501, options [nop,nop,TS val 371325006 ecr 30071274
32], length 0
13:28:46.440763 IP 72.21.91.29.http > router.49938: Flags [I], ack 863, win 133, options [nop,nop,TS val 3007137448 ecr 37130576
2], length 0
13:28:46.600900 IP router.49936 > 72.21.91.29.http: Flags [I], ack 6105, win 501, options [nop,nop,TS val 371326062 ecr 98322219
6], length 0
13:28:46.696364 IP 72.21.91.29.http > router.49936: Flags [I], ack 3080, win 302, options [nop,nop,TS val 983232436 ecr 37130575
4], length 0
^C
210 packets captured
210 packets received by filter
0 packets dropped by kernel
[root@router etc]#

```

Figure 4 tcpdump port 80

When a specific interface is not assigned, tcp dump reads the lowest interface, in these cases, eth0. Port 22 was used for watching ssh traffic and port 80 was watched for http traffic.

When using PuTTY to go over to the WebServerVM, the results did not show up on the server.



```

root@WebServerVM:~
login as: root
root@192.168.11.15's password:
Last login: Fri Apr 3 16:07:48 2020 from gateway
[root@WebServerVM ~]# ls
anaconda-ks.cfg
[root@WebServerVM ~]#

```

Figure 5 WebServerVM doesn't show on results for ssh

That is because the SSH connection did not pass through the eth0 interface.

An alternate way to scan for SSH is with this command line, which looks for all SSH on any port.

```
tcpdump 'tcp[(tcp[12]>>2):4] = 0x5353482D'
```

```
210 packets captured
210 packets received by filter
0 packets dropped by kernel
[root@router etc]# tcpdump 'tcp[(tcp[12]>>2):4]=0x5353482D'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:41:45.085971 IP 4.26.24.234.ssh > router.45273: Flags [P.], seq 2919790516:2919790535, ack 1641418739, win 4128, length 19
13:41:45.157935 IP router.45273 > 4.26.24.234.ssh: Flags [P.], seq 1:29, ack 19, win 64221, length 28
^[[A^[[B^[[C
2 packets captured
```

An alternate way to scan for HTTP traffic is with this command line which looks for all HTTP traffic on any port.

```
tcpdump -vAIs0
```

```
14:36:48.224922 IP (tos 0x0, ttl 58, id 60686, offset 0, flags [DF], proto TCP (6), length 40)
 104.17.68.176.https > router.57724: Flags [I.], cksum 0x5927 (correct), ack 46, win 68, length 0
E..(..@...h.D....i..i...G.P..DY".....
14:36:48.224990 IP (tos 0x0, ttl 58, id 60687, offset 0, flags [DF], proto TCP (6), length 86)
 104.17.68.176.https > router.57724: Flags [P.], cksum 0x5732 (correct), seq 1:47, ack 46, win 68, length 46
E..U..@...h.D....i..i...G.P..DW2.....).....HBI..{"-.....7{...^...$...
14:36:48.225136 IP (tos 0x0, ttl 63, id 61788, offset 0, flags [DF], proto TCP (6), length 40)
 router.57724 > 104.17.68.176.https: Flags [I.], cksum 0x4b0d (incorrect -> 0x5748), ack 47, win 501, length 0
E..(..@?...h.D..i...G...i.P...K...
14:36:48.229917 IP (tos 0x0, ttl 121, id 41515, offset 0, flags [none], proto UDP (17), length 135)
 dns.google.domain > router.40571: 7035 NXDomain 0/1/0 (107)
E....+..y.....5.f.s...f.....204.212.17.104.in-addr.arpa.....2.cruz.ns
cloudflare.com.dns.Ay2.....'.....:.....
14:36:48.241212 IP (tos 0x0, ttl 64, id 53289, offset 0, flags [DF], proto UDP (17), length 72)
 router.41160 > dns.google.domain: 23555+ PTR? 176.68.17.104.in-addr.arpa. (44)
E..H..@..@.....5.4..N.....176.68.17.104.in-addr.arpa.....
14:36:48.264987 IP (tos 0x0, ttl 121, id 57164, offset 0, flags [none], proto UDP (17), length 134)
 dns.google.domain > router.41160: 23555 NXDomain 0/1/0 (106)
E....L..y.....5...r.T.....176.68.17.104.in-addr.arpa.....1.2.cruz.ns
cloudflare.com.dns.Ay2.....'.....:.....
^C
331 packets captured
487 packets received by filter
156 packets dropped by kernel
[root@router etc]#
```

## Argus

Since the router has a broken argus, this line was used to try to repair and replace what is missing:

```
Yum install gcc make bison libpcap libpcap-devel readline-devel flex wget
```

When it installs correctly, it will say complete and show what was installed.

```

Running transaction test
Transaction test succeeded
Running transaction
Installing : mpfr-3.1.1-4.el7.x86_64 1/13
Installing : libmpc-1.0.1-3.el7.x86_64 2/13
Installing : m4-1.4.16-10.el7.x86_64 3/13
Installing : cpp-4.8.5-39.el7.x86_64 4/13
Installing : kernel-headers-3.10.0-1062.18.1.el7.x86_64 5/13
Installing : glibc-headers-2.17-292.el7.x86_64 6/13
Installing : glibc-devel-2.17-292.el7.x86_64 7/13
Installing : ncurses-devel-5.9-14.20130511.el7_4.x86_64 8/13
Installing : readline-devel-6.2-11.el7.x86_64 9/13
Installing : gcc-4.8.5-39.el7.x86_64 10/13
Installing : bison-3.0.4-2.el7.x86_64 11/13
Installing : flex-2.5.37-6.el7.x86_64 12/13
Installing : 14:libpcap-devel-1.5.3-11.el7.x86_64 13/13
Verifying : glibc-devel-2.17-292.el7.x86_64 1/13
Verifying : bison-3.0.4-2.el7.x86_64 2/13
Verifying : cpp-4.8.5-39.el7.x86_64 3/13
Verifying : mpfr-3.1.1-4.el7.x86_64 4/13
Verifying : ncurses-devel-5.9-14.20130511.el7_4.x86_64 5/13
Verifying : flex-2.5.37-6.el7.x86_64 6/13
Verifying : libmpc-1.0.1-3.el7.x86_64 7/13
Verifying : 14:libpcap-devel-1.5.3-11.el7.x86_64 8/13
Verifying : m4-1.4.16-10.el7.x86_64 9/13
Verifying : gcc-4.8.5-39.el7.x86_64 10/13
Verifying : readline-devel-6.2-11.el7.x86_64 11/13
Verifying : glibc-headers-2.17-292.el7.x86_64 12/13
Verifying : kernel-headers-3.10.0-1062.18.1.el7.x86_64 13/13

Installed:
  bison.x86_64 0:3.0.4-2.el7 flex.x86_64 0:2.5.37-6.el7 gcc.x86_64 0:4.8.5-39.el7 libpcap-devel.x86_64 14:1.5.3-11.el7
  readline-devel.x86_64 0:6.2-11.el7

Dependency Installed:
  cpp.x86_64 0:4.8.5-39.el7 glibc-devel.x86_64 0:2.17-292.el7 glibc-headers.x86_64 0:2.17-292.el7
  kernel-headers.x86_64 0:3.10.0-1062.18.1.el7 libmpc.x86_64 0:1.0.1-3.el7 m4.x86_64 0:1.4.16-10.el7
  mpfr.x86_64 0:3.1.1-4.el7 ncurses-devel.x86_64 0:5.9-14.20130511.el7_4

Complete!

```

Figure 6 argus complete install

The next steps were easy to follow, but a couple mistakes were made along the way which will be described after the errors came up in the return.

Source code packages for argus and argus clients are downloaded:

```
cd /usr/src
```

```
wget http://qosient.com/argus/src/argus-3.0.8.2.tar.gz
```

```
wget http://qosient.com/argus/src/argus-clients-3.0.8.2.tar.gz
```

```
tar -xaf argus-3.0.8.2.tar.gz
```

```
tar -xaf argus-clients-3.0.8.2.tar.gz
```

Argus is compiled:

```
cd /usr/src/argus-3.0.8.2
```

```
./configure
```

```
make && make install
```

The argus-clients packages are built:

```
cd /usr/src/argus-clients-3.0.8.2
```

```
./configure
```

```
make && make install
```

Argus service is configured by making a file in vi, the contents are below:

```
vi /etc/argus.conf
```

```
ARGUS_FLOW_TYPE="Bidirectional"
```

```
ARGUS_FLOW_KEY="CLASSIC_5_TUPLE"
```

```
ARGUS_DAEMON=yes
```

```
ARGUS_MONITOR_ID="eth1"
```

```
ARGUS_ACCESS_PORT=561
```

```
ARGUS_INTERFACE=eth1
```

```
ARGUS_SETUSER_ID=argus
```

```
ARGUS_SETGROUP_ID=argus
```

```
ARGUS_OUTPUT_FILE=/var/lib/argus/argus.out
```

```
ARGUS_FLOW_STATUS_INTERVAL=5
```

```
ARGUS_MAR_STATUS_INTERVAL=60
```

Argus user and group are created:

```
echo "argus:x:6000:6000:Argus:/home/argus:/sbin/nologin" >> /etc/passwd
```

```
echo "argus:x:6000:" >> /etc/group
```

Permissions are set on the output directory to read for all users except the owner:

```
mkdir /var/lib/argus
```

```
touch /var/lib/argus/argus.out
```

```
chown -R argus:argus /var/lib/argus
```

The Argus service is started:

```
/usr/local/sbin/argus -F /etc/argus.conf
```

Verification is needed that the service is running. This is done through netstat pulling up all the numbers that are running and grep finding the correct one on port 561.

```
netstat -an | grep 561
```

The first time this was ran, an error showed up, shown below.

```
"/etc/argus.conf" 11L, 343C written
[root@router argus-clients-3.0.8.21# echo "argus:x:6000:6000:Argus:/home/argus:/sbin/nologin" >> /etc/passwd
[root@router argus-clients-3.0.8.21# echo "argus:x:6000:" >> /etc/group
[root@router argus-clients-3.0.8.21# mkdir /var/lib/argus
mkdir: cannot create directory '/var/lib/argus': File exists
[root@router argus-clients-3.0.8.21# touch /var/lib/argus/argus.out
[root@router argus-clients-3.0.8.21# chown -R argus:argus /var/lib/argus
[root@router argus-clients-3.0.8.21# /usr/local/sbin/argus -F /etc/argus.conf
ArgusAlert: 13 Apr 20 17:12:50.438853 started
ArgusWarning: 13 Apr 20 17:12:50.444403 ArgusOpenInterface Interface: SIOCGIFHWADDR: No such device
ArgusWarning: 13 Apr 20 17:12:50.448759 ArgusOpenInterface LAN: SIOCGIFHWADDR: No such device
ArgusWarning: 13 Apr 20 17:12:50.453160 ArgusOpenInterface eth1 #Private: SIOCGIFHWADDR: No such device
[root@router argus-clients-3.0.8.21#
```

Figure 7 Argus Issues

At this point argus is not working. The clue is that there is no such device, but it has been told the device to use in the /etc/argus.conf file. What happened was there were two different typos, one was that the word DAEMON was spelled with the a and e switched, and on the ARGUS\_SETGROUP\_ID=argus there was a – instead of a \_. These two mistakes were difficult to see so that took time to find. During that time a great website was found that shows why this file looks and works like it does. It is part of the linux man pages and is found here:

<https://www.systutorials.com/docs/linux/man/5-argus.conf/>

Another set of mistakes was made because of trying the process more than once to find the errors. As found out in past labs, often trying commands again will create extra lines in files that need to be erased. There were extra lines in the /etc/passwd file as well as the /etc/group

file. Once those were erased down to one line as was proper, everything works properly when argus was tested using netstat:

```
Netstat -an | grep 561
```

```

"/etc/argus.conf" 11L, 350C written
[root@router /]# /usr/local/sbin/argus -F /etc/argus.conf
argus[5263]: 13 Apr 20 23:25:15.130432 started
[root@router /]# netstat -an | grep 561
tcp        0      0 0.0.0.0:561          0.0.0.0:*           LISTEN
[root@router /]#

```

Figure 8 Argus running correctly

The next command uses ratop and from it the network flow can be seen in real time as it passes through the router.

```
Ratop -S 127.0.0.1:561
```

At first there is no traffic so firefox is started again on the Kali VM.

```

23:57:13.662084 e      tcp  192.168.11.10.42222 -> 13.107.246.10.https 29 10086 CON
23:57:03.540439 e      tcp  192.168.11.10.38788 -> 54.149.124.142.https 27 6632 CON
23:57:03.521607 e      tcp  192.168.11.10.50274 -> 72.21.91.29.http 27 5488 CON
23:57:03.521581 e      tcp  192.168.11.10.50272 -> 72.21.91.29.http 26 4191 CON
23:57:29.076857 e      tcp  192.168.11.10.42238366 -> 13.107.272.217.10.67 25 8275 CON
23:57:45.203197 e      tcp  192.168.11.10.32920 -> 172.217.9.234.https 25 7280 CON
23:58:05.150416 e      tcp  192.168.11.10.52584 -> 209.85.201.154.https 24 6812 CON
23:57:13.083938 e      tcp  192.168.11.10.54742742 -> 151.1151.101.126.109 24 11397 CON
23:58:04.173002 e      tcp  192.168.11.10.40550006 -> 192.96.16.29.76tps 23 6098 RST
23:57:05.532346 e      tcp  192.168.11.10.50280 -> 72.21.91.29.http 23 2764 CON
23:57:13.883987 e      tcp  192.168.11.10.42744 -> 151.101.126.109.https 22 9492 CON
23:57:03.114664 e      tcp  192.168.11.10.52016 -> 23.40.62.19.http 22 2812 CON
23:57:13.261371 e      tcp  192.168.11.10.34358 -> 192.124.249.36.http 22 4178 FIN
23:57:13.261341 e      tcp  192.168.11.10.34356 -> 192.124.249.36.http 22 4178 FIN
23:57:13.261284 e      tcp  192.168.11.10.34354 -> 192.124.249.36.http 22 4178 FIN
23:57:13.261398 e      tcp  192.168.11.10.34362 -> 192.124.249.36.http 22 4178 FIN
23:57:13.261382 e      tcp  192.168.11.10.34360 -> 192.124.249.36.http 22 4178 FIN
23:58:04.173855 e      tcp  192.168.11.10.50090 -> 96.16.29.76.https 21 5966 RST
23:58:04.384849 e      tcp  192.168.11.10.49950008 -> 31.13.71.7.https 21 5611 RST
23:58:06.139911 e      tcp  192.168.11.10.50012 -> 31.13.71.7.https 21 5611 RST
23:57:45.735395 e      tcp  192.168.11.10.51210 -> 172.217.12.206.https 19 5310 RST
23:57:14.169426 e      tcp  192.168.11.10.51260962 -> 172.35.190.12.249tps 19 6033 CON
23:58:04.318525 e      tcp  192.168.11.10.51233996 -> 172.13.226.31.75tps 19 7061 CON
ProcessQueue 0 0isplayQueue 101 TotalRecords 345 Rate 4.5791 rps Status Active
[root@router /]#


```

Figure 9 ratop

Ratop provides information like Sending and Destination IP addresses, direction the information is going, time, TCP or UDP, destination port, total bytes, total packets and the state. It is also in real time where it moves as quickly as the internet browser page does. The output is different from tcpdump as tcpdump will show a lot of lines for something like connection setup

and data transfer where ratop will consolidate that information into one line and give the duration of time it took for the whole transaction. This is especially helpful to answer questions like “who is talking to whom, how often, is one address sending all the traffic, are they doing the bad thing?” (Auditing network activity, 2012) Being able to ask and answer these questions is very helpful to a network security professional or network administrator.

Reading through everything that Argus can do, especially as a beginner with it, can be a large task. A website that helped was openargus.org, specifically [openargus.org/oldsite/ra.core.examples.shtml](http://openargus.org/oldsite/ra.core.examples.shtml) . Below is a pic of some of the argus clients that help with data flow. Once this list was found, choosing two clients to use with argus was much easier.



- Using Argus
- Getting Argus
- Argus Wiki
- Development
- Documentation
- Publications
- Support
- Links
- News

**Ra Core Client Examples**

The argus-clients package provides a set of core client programs that provide the basic functions needed to use argus flow data. This includes printing, processing, sorting, aggregating, tallying, collecting, distributing, archiving, and anonymizing data. Here we provide basic examples of how to use these utilities; ra, rabins, racluster, racount, radium, ranonymize, rasort and rasplit. For these examples, we use standard sets of [argus data](#).

<a href="#">ra</a>	Basic argus record reading and printing and storing
<a href="#">rabins</a>	Align argus data to time based bins.
<a href="#">racluster</a>	Argus data aggregation
<a href="#">racount</a>	Tally various aspects of an argus stream.
<a href="#">radium</a>	Argus record collection and distribution.
<a href="#">ranonymize</a>	Anonymization of argus data.
<a href="#">rasort</a>	Argus file or stream sorting.
<a href="#">rasplit</a>	Improve and document what we've got.

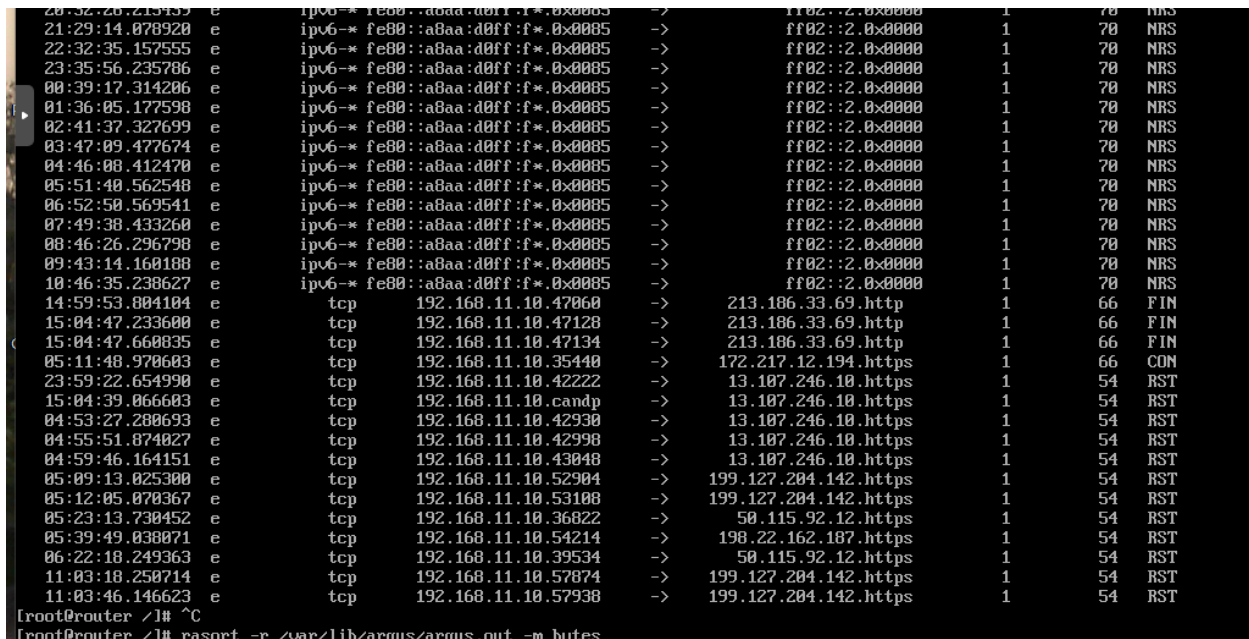
Each of these core programs provide a basic set of features that are needed to get utility from argus based flow data.

At this point the process to acquire the clients like rasort, racluster and rafilteraddr was unclear.

The clients were already installed, however, so once the work has been done to this point of the lab, the clients are already installed and ready to work with .To begin client work by working off

the man pages, `rasort` was used as it provided very simple directions and gave a focused look at where all the information was at. For example, when wanting to sort by bytes of the total transaction, use:

```
rasort -r /var/lib/argus/argus.out -m bytes
```



```

28:32:28.213153 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
21:29:14.878928 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
22:32:35.157555 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
23:35:56.235786 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
00:39:17.314206 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
01:36:05.177598 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
02:41:37.327699 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
03:47:09.477674 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
04:46:08.412470 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
05:51:48.562548 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
06:52:50.569541 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
07:49:38.433260 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
08:46:26.296798 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
09:43:14.160188 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
10:46:35.238627 e      ipv6-* fe80::a8aa:d0ff:f*.0x0005 -> ff02::2.0x0000 1 70 NRS
14:59:53.804104 e      tcp    192.168.11.10.47060 -> 213.186.33.69.http 1 66 FIN
15:04:47.233608 e      tcp    192.168.11.10.47128 -> 213.186.33.69.http 1 66 FIN
15:04:47.660835 e      tcp    192.168.11.10.47134 -> 213.186.33.69.http 1 66 FIN
05:11:48.970603 e      tcp    192.168.11.10.35440 -> 172.217.12.194.https 1 66 CON
23:59:22.654990 e      tcp    192.168.11.10.42222 -> 13.107.246.10.https 1 54 RST
15:04:39.066603 e      tcp    192.168.11.10.candp -> 13.107.246.10.https 1 54 RST
04:53:27.280693 e      tcp    192.168.11.10.42930 -> 13.107.246.10.https 1 54 RST
04:55:51.874827 e      tcp    192.168.11.10.42998 -> 13.107.246.10.https 1 54 RST
04:59:46.164151 e      tcp    192.168.11.10.43048 -> 13.107.246.10.https 1 54 RST
05:09:13.025300 e      tcp    192.168.11.10.52904 -> 199.127.204.142.https 1 54 RST
05:12:05.070367 e      tcp    192.168.11.10.53108 -> 199.127.204.142.https 1 54 RST
05:23:13.730452 e      tcp    192.168.11.10.36822 -> 50.115.92.12.https 1 54 RST
05:39:49.038071 e      tcp    192.168.11.10.54214 -> 198.22.162.187.https 1 54 RST
06:22:18.249363 e      tcp    192.168.11.10.39534 -> 50.115.92.12.https 1 54 RST
11:03:18.250714 e      tcp    192.168.11.10.57874 -> 199.127.204.142.https 1 54 RST
11:03:46.146623 e      tcp    192.168.11.10.57938 -> 199.127.204.142.https 1 54 RST
[root@router ~]# ^C
[root@router ~]# rasort -r /var/lib/argus/argus.out -m bytes

```

Figure 10 `rasort` sorting for bytes

When sorting for total transaction packets, use:

```
rasort -r /var/lib/argus/argus.out -m pkts
```



```

06:42:48.458958 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:42:58.698949 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:43:08.938938 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 6 2982 CON
06:43:23.530885 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:43:28.684884 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 2858 CON
06:43:38.890831 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 6 2982 CON
06:43:53.738746 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:03.978775 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:14.218749 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:24.458728 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:34.698665 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:44.938761 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:55.178664 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:45:05.418557 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:45:15.658504 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 6 2982 CON
06:45:26.410602 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:45:34.654484 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 2858 CON
06:45:44.842472 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:45:55.082499 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:05.322570 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:15.562456 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:25.802432 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:36.042394 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:46.282374 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:52.334915 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 2858 CON
06:47:02.666382 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:47:10.191868 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 2858 CON
06:47:20.586375 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 1661 CON
06:47:25.682819 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 1321 CON
06:47:35.690258 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:47:45.930417 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:47:56.170343 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:06.410311 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:16.650189 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:26.890215 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:37.130181 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:47.370112 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:57.610070 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:49:07.850189 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:49:18.090072 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 8 581 FIN
root@router /# rasort -r /var/lib/argus/argus.out -m sport

```

Figure 11 rasort sorting for packet count

Below the source port was sorted by using:

```
rasort -r /var/lib/argus/argus.out -m sport
```

```

06:44:24.458728 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:34.698665 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:44.938761 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:44:55.178664 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:45:05.418557 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:45:15.658504 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 6 2982 CON
06:45:26.410602 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:45:34.654484 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 2850 CON
06:45:44.842472 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:45:55.082499 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:05.322570 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:15.562456 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:25.082432 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:36.042394 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:46.282374 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:46:52.334915 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 2850 CON
06:47:02.666382 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:47:10.191868 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 2850 CON
06:47:20.586375 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 4 1661 CON
06:47:25.682819 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 1321 CON
06:47:35.690258 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:47:45.930417 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:47:56.170343 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:06.410311 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:16.650189 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:26.090215 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:37.130181 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:47.370112 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:48:57.610070 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:49:07.850189 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:49:18.090072 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 8 581 FIN
[root@router /]# rasort -r /var/lib/argus/argus.out -m sport

```

Figure 12 rasort for source port

Finally, the oui of the source MAC address was sorted for

```
rasort -r /var/lib/argus/argus.out -m soui
```

```

06:17:17.742621 e tcp 192.168.11.10.40498 -> 23.239.15.111.https 2 132 CON
06:17:17.806697 e tcp 192.168.11.10.59132 -> 199.127.204.100.https 2 132 CON
06:17:17.822658 e tcp 192.168.11.10.51774 -> 96.16.29.69.https 2 132 CON
06:17:17.838540 e tcp 192.168.11.10.58138 -> 199.127.204.100.https 2 132 CON
06:17:17.838569 e tcp 192.168.11.10.60990 -> 192.35.249.124.https 2 132 CON
06:17:18.000030 e tcp 192.168.11.10.46932 -> 68.67.179.87.https 7 500 RST
06:17:18.019102 e tcp 192.168.11.10.46934 -> 68.67.179.87.https 7 512 RST
06:17:18.034681 e tcp 192.168.11.10.46930 -> 68.67.179.87.https 7 500 RST
06:17:18.086289 e tcp 192.168.11.10.46936 -> 68.67.179.87.https 7 500 RST
06:17:18.102562 e tcp 192.168.11.10.55836 -> 198.22.162.187.https 7 524 FIN
06:17:18.626067 e tcp 192.168.11.10.55822 -> 198.22.162.187.https 4 2397 CON
06:17:19.000777 e tcp 192.168.11.10.50790 -> 38.134.110.188.https 7 440 RST
06:17:19.034597 e tcp 192.168.11.10.50788 -> 38.134.110.188.https 7 440 RST
06:17:22.327849 e tcp 192.168.11.10.42890 -> 23.92.190.69.https 7 486 RST
06:17:22.371145 e tcp 192.168.11.10.42894 -> 23.92.190.69.https 4 264 RST
06:17:23.122834 e tcp 192.168.11.10.55836 -> 198.22.162.187.https 2 108 RST
06:17:23.612242 e tcp 192.168.11.10.55874 -> 198.22.162.187.https 14 4388 CON
06:17:23.625895 e tcp 192.168.11.10.46600 -> 54.209.83.202.https 7 2213 CON
06:17:23.626644 e udp 192.168.11.10.45759 <-> 8.8.4.4.domain 2 402 CON
06:17:23.626675 e udp 192.168.11.10.45759 <-> 8.8.4.4.domain 2 337 CON
06:17:23.628146 e udp 192.168.11.10.59020 <-> 8.8.4.4.domain 2 156 CON
06:17:23.628171 e udp 192.168.11.10.59020 <-> 8.8.4.4.domain 2 205 CON
06:17:23.628247 e tcp 192.168.11.10.58138 -> 199.127.204.100.https 4 2035 CON
06:17:23.630537 e udp 192.168.11.10.47504 <-> 8.8.4.4.domain 2 271 CON
06:17:23.630573 e udp 192.168.11.10.47504 <-> 8.8.4.4.domain 2 289 CON
[root@router /]# rasort -r /var/lib/argus/argus.out -m soui

```

Figure 13 rasort for oui of the source MAC address

The next client used is a filter for IP addresses. This client requires another file to be made that has the IP addresses in it that are wanted to be searched for. This file can be anywhere

on the computer but needs to be input into the command line, along with the output argus file.

For the first try, just the standard IP address was used of the KaliVM machine, 192.168.11.10

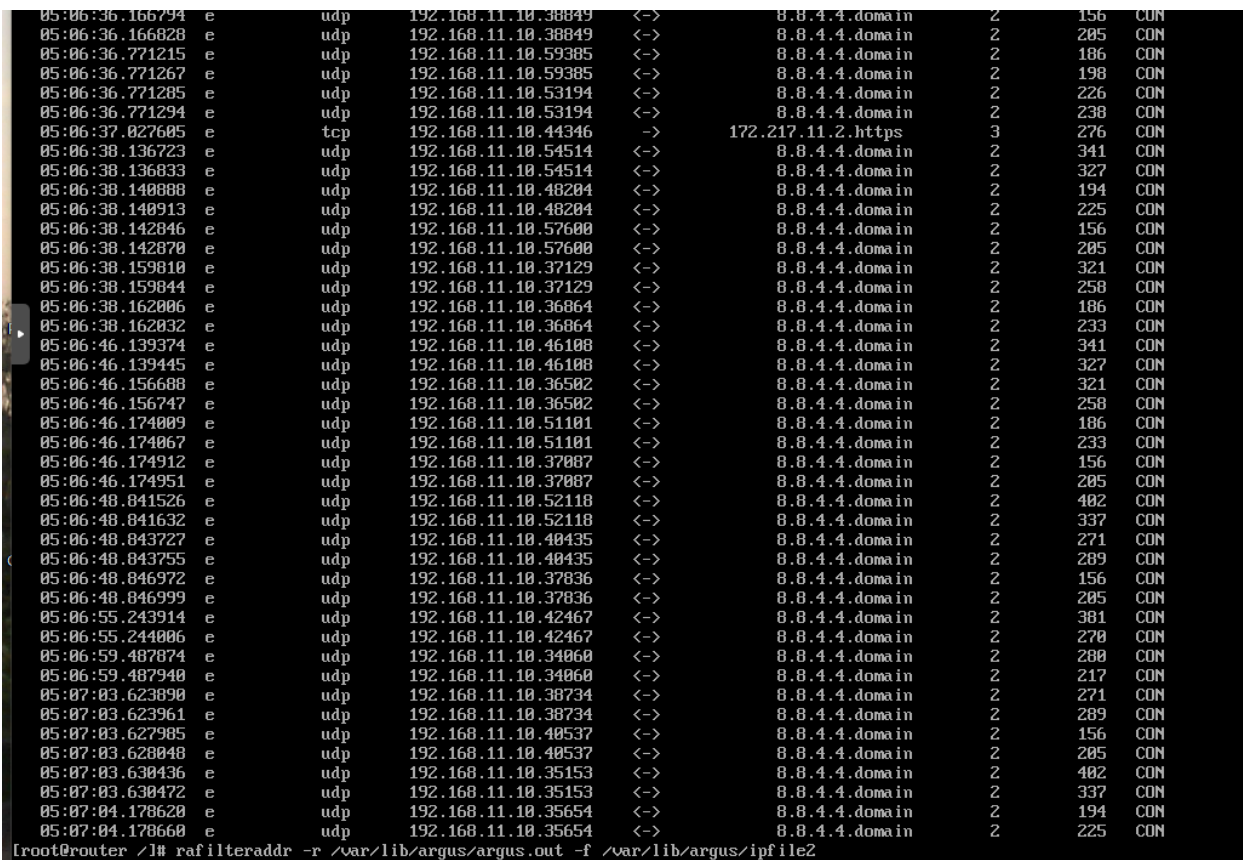
```
rafilteraddr -r /var/lib/argus/argus.out -f /var/lib/argus/ipfile
```

11:23:32.168788	e	tcp	192.168.11.10.45682	->	38.134.110.237.https	9	548	RST
11:23:32.192854	e	tcp	192.168.11.10.58418	->	198.22.162.187.https	9	632	RST
11:23:33.728932	e	tcp	192.168.11.10.58352	->	96.6.24.95.https	8	578	RST
11:23:33.770808	e	udp	192.168.11.10.56833	<->	8.8.4.4.domain	2	280	CON
11:23:33.770808	e	udp	192.168.11.10.56833	<->	8.8.4.4.domain	2	217	CON
11:23:33.798742	e	tcp	192.168.11.10.56942	->	198.22.162.187.https	8	3351	CON
11:23:36.958887	e	tcp	192.168.11.10.48808	->	104.129.131.89.https	61	136741	CON
11:23:40.771846	e	tcp	192.168.11.10.36862	->	172.217.12.130.https	3	276	CON
11:23:40.771115	e	tcp	192.168.11.10.42216	->	172.217.12.194.https	3	276	CON
11:23:40.771148	e	tcp	192.168.11.10.36532	->	172.217.12.166.https	3	276	CON
11:23:41.664949	e	tcp	192.168.11.10.47936	->	198.54.12.97.https	6	2982	CON
11:23:41.665806	e	tcp	192.168.11.10.68278	->	199.127.204.100.https	6	2169	CON
11:23:41.665812	e	tcp	192.168.11.10.35374	->	192.35.249.124.https	6	2982	CON
11:23:43.817847	e	udp	192.168.11.10.58813	<->	8.8.4.4.domain	2	381	CON
11:23:43.817950	e	udp	192.168.11.10.58813	<->	8.8.4.4.domain	2	270	CON
11:23:43.817969	e	tcp	192.168.11.10.48808	->	104.129.131.89.https	57	137605	CON
11:23:43.771664	e	tcp	192.168.11.10.56942	->	198.22.162.187.https	17	18594	CON
11:23:45.195927	e	udp	192.168.11.10.47926	<->	8.8.4.4.domain	2	482	CON

Figure 14 rfilteraddr for IP address 192.168.11.10

Since there are a few different IP addresses in the above pic, three were chosen and a new file (ipfile2) was created to filter from using IP addresses 8.8.4.4, 172.217.11.2 and 8.8.8.8.

```
rafilteraddr -r /var/lib/argus/argus.out -f /var/lib/argus/ipfile2
```



05:06:36.166794	e	udp	192.168.11.10.38849	<->	8.8.4.4.domain	2	156	CON
05:06:36.166828	e	udp	192.168.11.10.38849	<->	8.8.4.4.domain	2	205	CON
05:06:36.771215	e	udp	192.168.11.10.59385	<->	8.8.4.4.domain	2	186	CON
05:06:36.771267	e	udp	192.168.11.10.59385	<->	8.8.4.4.domain	2	198	CON
05:06:36.771285	e	udp	192.168.11.10.53194	<->	8.8.4.4.domain	2	226	CON
05:06:36.771294	e	udp	192.168.11.10.53194	<->	8.8.4.4.domain	2	238	CON
05:06:37.027605	e	tcp	192.168.11.10.44346	->	172.217.11.2.https	3	276	CON
05:06:38.136723	e	udp	192.168.11.10.54514	<->	8.8.4.4.domain	2	341	CON
05:06:38.136833	e	udp	192.168.11.10.54514	<->	8.8.4.4.domain	2	327	CON
05:06:38.140888	e	udp	192.168.11.10.48204	<->	8.8.4.4.domain	2	194	CON
05:06:38.140913	e	udp	192.168.11.10.48204	<->	8.8.4.4.domain	2	225	CON
05:06:38.142846	e	udp	192.168.11.10.57600	<->	8.8.4.4.domain	2	156	CON
05:06:38.142870	e	udp	192.168.11.10.57600	<->	8.8.4.4.domain	2	205	CON
05:06:38.159810	e	udp	192.168.11.10.37129	<->	8.8.4.4.domain	2	321	CON
05:06:38.159844	e	udp	192.168.11.10.37129	<->	8.8.4.4.domain	2	258	CON
05:06:38.162806	e	udp	192.168.11.10.36864	<->	8.8.4.4.domain	2	186	CON
05:06:38.162832	e	udp	192.168.11.10.36864	<->	8.8.4.4.domain	2	233	CON
05:06:46.139374	e	udp	192.168.11.10.46188	<->	8.8.4.4.domain	2	341	CON
05:06:46.139445	e	udp	192.168.11.10.46188	<->	8.8.4.4.domain	2	327	CON
05:06:46.156688	e	udp	192.168.11.10.36502	<->	8.8.4.4.domain	2	321	CON
05:06:46.156747	e	udp	192.168.11.10.36502	<->	8.8.4.4.domain	2	258	CON
05:06:46.174809	e	udp	192.168.11.10.51101	<->	8.8.4.4.domain	2	186	CON
05:06:46.174867	e	udp	192.168.11.10.51101	<->	8.8.4.4.domain	2	233	CON
05:06:46.174912	e	udp	192.168.11.10.37087	<->	8.8.4.4.domain	2	156	CON
05:06:46.174951	e	udp	192.168.11.10.37087	<->	8.8.4.4.domain	2	205	CON
05:06:48.841526	e	udp	192.168.11.10.52118	<->	8.8.4.4.domain	2	402	CON
05:06:48.841632	e	udp	192.168.11.10.52118	<->	8.8.4.4.domain	2	337	CON
05:06:48.843727	e	udp	192.168.11.10.40435	<->	8.8.4.4.domain	2	271	CON
05:06:48.843755	e	udp	192.168.11.10.40435	<->	8.8.4.4.domain	2	289	CON
05:06:48.846972	e	udp	192.168.11.10.37836	<->	8.8.4.4.domain	2	156	CON
05:06:48.846999	e	udp	192.168.11.10.37836	<->	8.8.4.4.domain	2	205	CON
05:06:55.243914	e	udp	192.168.11.10.42467	<->	8.8.4.4.domain	2	381	CON
05:06:55.244006	e	udp	192.168.11.10.42467	<->	8.8.4.4.domain	2	270	CON
05:06:59.487874	e	udp	192.168.11.10.34060	<->	8.8.4.4.domain	2	280	CON
05:06:59.487940	e	udp	192.168.11.10.34060	<->	8.8.4.4.domain	2	217	CON
05:07:03.623890	e	udp	192.168.11.10.38734	<->	8.8.4.4.domain	2	271	CON
05:07:03.623961	e	udp	192.168.11.10.38734	<->	8.8.4.4.domain	2	289	CON
05:07:03.627985	e	udp	192.168.11.10.40537	<->	8.8.4.4.domain	2	156	CON
05:07:03.628048	e	udp	192.168.11.10.40537	<->	8.8.4.4.domain	2	205	CON
05:07:03.630436	e	udp	192.168.11.10.35153	<->	8.8.4.4.domain	2	402	CON
05:07:03.630472	e	udp	192.168.11.10.35153	<->	8.8.4.4.domain	2	337	CON
05:07:04.178620	e	udp	192.168.11.10.35654	<->	8.8.4.4.domain	2	194	CON
05:07:04.178660	e	udp	192.168.11.10.35654	<->	8.8.4.4.domain	2	225	CON

```
[root@router /]# rafilteraddr -r /var/lib/argus/argus.out -f /var/lib/argus/ipfile2
```

Figure 15 rafilteraddr IP address 2

In the right column of IP addresses, 8.8.4.4 is appearing the most but the filter is working properly as the address 172.217.11.2 can be seen toward the top. This feature could be useful when searching for specific known malicious IP addresses, say from a botnet server or a known IP address that has repeatedly attacked a system, to see if it is still happening or to see if mitigation may need to take place or company emails sent out, as in the case of botnet spam.

## Conclusion.

This lab was successful at teaching how to set up tcpdump and argus on a CentOS 7 server. It also gave more clarity on man pages, how to read them, what to look for in the details and what a program client needs to work properly. Using the tools rasort and rafilteraddr helped the student learn more about what a client program is and how they are attached to a main

program like argus and what they help do. The rasort program was a standard sorting tool that shows more information than just filtering out for one item. This could be useful when looking for malicious activity as hackers can often change one or two details to fool servers into thinking they are legitimate. The rfilteraddr is a filter that scans for specific IP addresses that have been put into a file. When these addresses appear, they are shown on the screen. This would be helpful for security as when malicious IP addresses have been previously been identified, they can be put into the list file and any attempts by them to infiltrate the system could be seen clearly as soon as the attack begins. Overall this lab was an excellent learning experience and a great source for future traffic monitoring.

## References

- Abrams, Lawrence. (November 29, 2019). *Kali Linux adds 'undercover' mode to impersonate Windows 10*. BleepingComputer. Retrieved from <https://www.bleepingcomputer.com/news/security/kali-linux-adds-undercover-mode-to-impersonate-windows-10/>
- Argus. (2020). Definition from openargus.org. Retrieved from <https://openargus.org/>
- Auditing network activity. (2012). Argus. Retrieved from <https://qosient.com/argus/gettingstarted.shtml>
- Centos Blog (2020). *What is Centos?* Retrieved from <https://www.centosblog.com/what-is-centos/>
- Cheng, Simon M.C. (October 27, 2014). *Basic concept of ProxMox Virtual Environment*. Packt. Retrieved from <https://hub.packtpub.com/basic-concepts-proxmox-virtual-environment/>
- Elwood. (November 26, 2019). *Kali Linux 2019.4 release*. Kali Linux News. Retrieved from <https://www.kali.org/news/kali-linux-2019-4-release/>
- Gerardi, Ricardo. (October 10, 2018). *An introduction to using tcpdump at the Linux command line*. Retrieved from <https://opensource.com/article/18/10/introduction-tcpdump>
- ProxMox. (April 11, 2019). ProxMox.com Retrieved from <https://www.proxmox.com/en/news/press-releases/proxmox-ve-5-4>
- SSH(Secure Shell). (2020). SSH.com. Retrieved from <https://www.ssh.com/ssh>
- What is KVM? (2020) RedHat.com Retrieved from <https://www.redhat.com/en/topics/virtualization/what-is-KVM>

## Lab Network Topology

Kali2020VM--VMWare ethernet adapter--Student HP----(internal –router—external)

192.168.22.136      192.168.22.1      10.0.0.17    10.0.0.1      192.168.104.161

/

/

WWW

|

Gateway 10.42.0.1

|

chewy 10.42.0.31/16

|

darknet 172.16.0.3/16

|

External 172.16.242.32

router

Internal 192.168.11.1

/

\

Kali VM 192.168.11.10

WebServerVM 192.168.11.15