Introduction to Kerberos Single Sign on realm

Aim

The aim of this lab is to introduce kerberos protocol for sso ticket generation, by simulating it on the lab system

Introduction and background

The main focus of this lab is to simulate a kerberos ticket generation for sso login between two parties. To complete this task, we create two virtual machines, one set up as the admin server/ Kerberos Key Distribution Center and the other as an example client. We then have the client request a ticket from the admin server and have it authenticate with the server.

Methods

To begin we create a new VM which will act as an example client. We do this by virtual box. The new VM is an exact copy of our original machine, which will be used as the admin server. We then install and configure our kerberos server. For our kerberos realm we use SENG360.COM. We then add principles to this server. This principle is allowed to access services across the kerberos sso realm. After we set up kerberos client software on the client VM. Finally, we request a key from the admin server using the principle we set up previously. This gives the client machine a ticket which the client can use to authenticate itself to other principles in the realm.

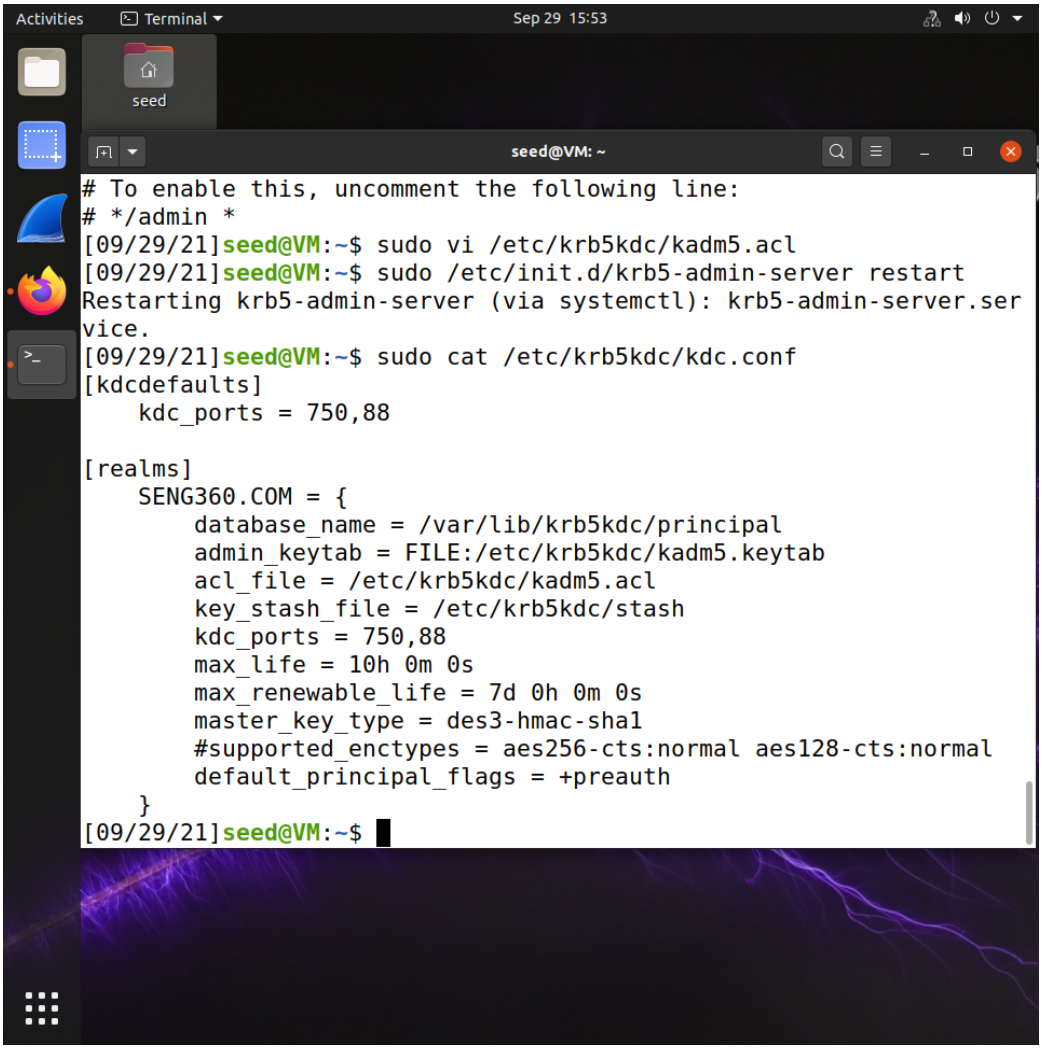
Results and Observations

After successfully setting up the Virtual machines we check the internal IP address of each machine.

```
[09/29/21]seed@VM:~$ ifconfig -a
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:5f:8f:50:a6  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.211  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::b13d:b2b4:187a:1aba  prefixlen 64  scopeid 0x20<link>
        ether 52:54:00:11:7f:08  txqueuelen 1000  (Ethernet)
        RX packets 928  bytes 134631 (134.6 KB)
        RX errors 0  dropped 37  overruns 0  frame 0
        TX packets 224  bytes 25560 (25.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 221  bytes 21050 (21.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
```

```
[09/29/21]seed@VM:~$ ifconfig -a
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.
255
        ether 02:42:79:26:c4:93  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.240  netmask 255.255.255.0  broadcast 192.16
8.1.255
        inet6 fe80::ea09:b4e3:c34b:f919  prefixlen 64  scopeid 0x20
<link>
        ether 52:54:00:95:bd:40  txqueuelen 1000  (Ethernet)
        RX packets 10120  bytes 7225451 (7.2 MB)
        RX errors 0  dropped 70  overruns 0  frame 0
        TX packets 5160  bytes 752368 (752.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
```

After configuration the admin server and looking through the contents we try to answer the following question



(Question Q1: What do the *max life* and *max_renewable_life* parameters mean?) (you may have to do some research)

Max life refers to the lifetime of the ticket after which the ticket is no longer valid and the client will need to generate a new ticket to authenticate themselves.

Max renewable life is the max time the client is allowed to renew the ticket. The client has to renew before the first expiration time.

On the client side, we request a key from the KDC. Once the client authenticates itself, we take a look at the ticket issued.

```
[09/29/21]seed@VM:~$ kinit alice@SENG360.COM
Password for alice@SENG360.COM:
[09/29/21]seed@VM:~$ klist -e
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: alice@SENG360.COM

Valid starting       Expires              Service principal
09/29/2021 16:01:03  09/30/2021 02:01:03  krbtgt/SENG360.COM@SENG360.COM
        renew until 09/30/2021 16:01:00, Etype (skey, tkt): aes256-cts-hmac-sha1
-96, aes256-cts-hmac-sha1-96
[09/29/21]seed@VM:~$
```

(Question Q2: How does the ticket start and expiration time relate to the settings you observed earlier?)

Initially, the ticket is only valid for the max time mentioned above. Here start time to expiration date equals the max time above. The expiration time changes if the client requests to renew the ticket. This will cause the expiration time to increase depending on the max_renewable_life value shown above.