flask-kerberos-module

▶ Table of Contents

About The project

This repository contains implemented solution for **kerberos authentification** in web context. A server expose files that are protected and shouldn't be accessible unless the user is authenticated. This project is implemented on **LINUX machines** using the **Ubuntu 20.04 LTS** distribution.

(back to top)

Built With

- Python, the widely used interpreted language.
- Flask, micro framework open-source de développement web en Python. the python micro-framework for wen developpement.

(back to top)

What is Kerberos

Kerberos is an AAA authentication protocol from the Massachusetts Institute of Technology (MIT) ”Athena” project. it is responsible for **authenticating, authorizing, and monitoring users** who want to access *resources and services* on your network thus the correspondance with the three-headed dog guardian of the gates of Hades in the greek mythology.
kerberos introduces the principle of **Single Sign-On (SSO)** . Thus with a single authentication, the user will have access to all the services of the network. it relies on a trusted third party to manage authentication, the **KDC (Key Distribution Center)**. All users and services on the network trust this third party.
Kerberos uses a ticket system to perform authentication and introduces **the principle of SSO (Single Sign On)**. The user authenticates on the KDC and then uses a ticket to authenticate on each requested service. (no password sending via network).

(back to top)

Getting Started

In order to run this project, we need to follow some few steps :

Prerequisites

- Make sure that you have a virtualization software.In this demo i used **Oracle VM VirtualBox** (Download Here).
- Make sure you have 2 linux machines with the **Ubuntu 20.04 LTS distribution** (Download Here).
- Make sure you have **python3** and **pip** on the server machine.

```
 sudo apt update
sudo apt install python3
sudo apt install python3-pip
```

Kerberos Configuration

1. Environnement

In order to proceed with the configurations we need to have a : - Domain name : "example.tn" - Realm : "EXAMPLE.TN" - Two machines :

| Machine Name | Machine IP | Sub-domain name | | :---: | :---: | :---: | | KDC | 192.168.56.110 | kdc.example.tn | | server | 192.168.56.111 | server.example.tn |

> machines IP's are just an example, use `hostname -I` to get each machine ip.
> All the configurations must be done in **root** mode, use `su -` to connect as root.

(back to top)

2. DNS (Domain name system)

Used to match domain name to their IP's.

```
nano /etc/hosts
```

and add *(for each machine)* :

```
 192.168.56.110    kdc.example.tn       kdc
192.168.56.111    server.example.tn    server
```

then set the **hostname** *(for each machine)* :

Machine Name	set new hostname
KDC	`hostnamectl set-hostname kdc.example.tn`
server	`hostnamectl set-hostname server.example.tn`

(back to top)

3. Time Synchronization

When the client obtains a ticket from Kerberos, it includes in its message the current time of day. One of the three parts of the response from Kerberos is a timestamp issued by the Kerberos server.

3.1. on the ***KDC*** install **ntp**:

```
apt install ntp
```

then edit the `/etc/ntp.conf` and add the lines below under the `# local users may interrogate the ntp server more closely` section:

```
 restrict 127.0.0.1
restrict ::1
restrict 192.168.56.110 mask 255.255.255.0
nomodify notrap
server 127.127.1.0 stratum 10
listen on *
```

3.2. on the ***server*** install **ntp** and **ntpdate**:

```
 apt install ntp
apt install ntpdate
```

then edit the `/etc/ntp.conf` and add the lines below under the `# Use Ubuntu's ntp server as a fallback` section:

```
 pool ntp.ubuntu.com
server 192.168.56.110
server obelix
```

3.3. Synchronize time by running the below command on the server machine:

```
ntpdate -dv 192.168.56.110
```

(back to top)

4. Configure KDC

4.1. We need to install **the packages** *krb5-kdc*, *krb5-admin-server* and *krb5-config* by running :

```
apt install krb5-kdc krb5-admin-server krb5-config
```

During installation you will be prompted to enter the *realm*, *kerberos server* and *administartive server* and it would be in order: | Prompt | value | | :---: | :---: | | Realm | EXAMPLE.TN | | Kerberos servers | kdc.example.tn | | Administrative Service | kdc.example.tn |

> Its capital sensitive.
> View kdc settings with `cat /etc/krb5kdc/kdc.conf`.
> The error **failed to start kerberos 5 ..** will not be a problem.

4.2 Now we need to add **kerberos database** where principals will be stored

```
krb5_newrealm
```

> You will be prompted to choose a password.

4.3 we will create an *admin principal* , a *host principal* and generate its keytab: - **principal:** a unique identity to which Kerberos can assign tickets. - **keytab:** stores long-term keys for one or more principals and allow server applications to accept authentications from clients, but can also be used to obtain initial credentials for client applications. run the following commands:

```
kadmin.local                              # login as local admin
addprinc root/admin                       # add admin principal
addprinc -randkey host/kdc.example.tn     # add host principal
ktadd host/kdc.example.tn                 # generate host principal keytab
```

> type `q` to exit.

4.3 Grant the **admin principal** all privileges by editing `/etc/krb5kdc/kadm5.acl`:

```
root/admin *                              # just uncomment the line
```

4.4 restart the kerberos service by running:

```
systemctl restart krb5-admin-server
systemctl status krb5-admin-server        # to check service status
```

(back to top)

5. Configure Server

5.1. We need to install **the packages** *krb5-user*, *libpam-krb5* and *libpam-ccreds* by running:

```
apt install krb5-user libpam-krb5 libpam-ccreds
```

During installation you will be prompted to enter the *realm*, *kerberos server* and *administartive server* and it would be in order: | Prompt | value | | :---: | :---: | | Realm | EXAMPLE.TN | | Kerberos servers | kdc.example.tn | | Administrative Service | kdc.example.tn |

> Its capital sensitive.
> View krb settings with `cat /etc/krb5.conf`.

5.2 we will create a *host principal* and generate its keytab by running:

```
kadmin                                       # login as admin (type your password)
addprinc -randkey host/server.example.tn     # add host principal
ktadd host/server.example.tn                 # generate host principal keytab
```

> type `q` to exit.

5.3 Add a test user and create a correspending principal by running:

```
useradd -m -s /bin/bash testUser
kadmin
addprinc testUser
```

> type `q` to exit.

(back to top)

Flask Configuration

We install flask nn the **server** machine by running:

```
pip install Flask
```

Then we need to install the **Flask_kerberos** module by running:

```
pip install Flask-Kerberos
```

> If an error occured run `apt install libkrb5-dev` then restart the Flask_kerberos module.
> If it still persists, check your gcc installation.

Then we need to install the **Flask_bootstrap** module by running:

```
pip install flask_bootstrap
```

(back to top)

Installation

1. Clone the repo in the server `console git clone https://github.com/hamza-mahjoub/flask-kerberos-module.git.`
2. Login as root by running `su -`.
3. set the **KRB5_KTNAME** variable that references the **keytab**:

```
export KRB5_KTNAME=/etc/krb5.keytab
```

> To visualize your keytab, run the **ktutil** as root.
>
> ```
>  ktutil
> ?                         # list all commands
> read_kt /etc/krb5.keytab  # read keytab file
> list                      # show principals
> ```

4. Turn all scripts executable by running `chmod 755 script_name.py`. > There are 5 scripts: app.py, check_route.py, negotiate.py, add_line.py, delete_line.py.

(back to top)

Usage

Before running the main app script you need to change hostname in each script to the one you chose

Script	line
app.py	99
negotiate.py	8
add_line.py	8
delete_line.py	8

run the main app script by the command `./app.py` as **root** and you should see in the terminal that the server is running.

```
# some informations
running on http://127.0.0.1:8080
```

Routes

The routes of the flask api.

Route	protected	description
/	✖	return a description about the project
/home	✔	return the list of files under the `files` directory
/path/path	✔	return the content of the selected file
/addline.py	✔	add any number or lines to a selected file
/deleteline/path/path	✔	delete a line from a selected file
/test/home	✖	same as `/home` but with no protection
/test/	✖	same as `/<path:path>` but with no protection

> The `/test` routes are for visualization on **firefox**

Scripts

They help creating **requests** to the routes without needing a front end. They are based on the python *requests module*.

Script	parameters	description

Script	parameters	description
check_route.py	route_name `(ex: home)`	check if a route is protected. if not return the response
negotiate.py	route_name `(ex: home)`	uses kerberos ticket to access a route and return response or return error
add_line.py	file_path list_lines `(ex: files/file1.txt hallo)`	add a line to a file, kerberos ticket must exists
delete_line.py	file_path line_number `(ex: files/file2.txt 2)`	delete a line from a file (start = 0), kerberos ticket must exists

Kerberos scenario

test a scenario by running the following commands:

```
 su testUser                   # login as the test user (use another shell then the one where the app is running)
./check_route /                # return response content and status code 200.
./check_route home             # return status code 401 : unauthorized and it asks for negotiation.
./negotiate.py home            # return a kerberos.GSSErro because it can't find kerberos credentials.
```

- We must generate a ticket so we use the command `kinit` . > To verify the ticket we can run `klist` . It shows that it is of type **krbtgt/..** (kerberos ticket granting ticket).
- **TGT**: a user authentication token issued by the Key Distribution Center (KDC) that is used to request access tokens from the Ticket Granting Service (TGS) for specific resources/systems joined to the domain. > To destroy the ticket we can run `kdestroy` .
- Now we try to negotiate

```
 ./negotiate.py home           # return response content (list of files) and status code 200.
```

> When we run `klist` . It shows that a **TGS** is created (kerberos ticket granting service). * **TGS**: provides tickets and Ticket Granting Tickets to the client systems. Ticket Granting Tickets contain the client ID, the client network address, the ticket validity period, and the Ticket Granting Server session key which is used to access service without password exchanging. let try adding lines and deleting them by running these commands:

```
 ./negotiate.py files/file1.txt                 # return content of file1 and status code 200.
./add_line.py files/file1.txt hallo there     # add two lines "hallo" and "there" to file1.txt.
./delete_line.py files/file1.txt 2            # delete the line of index 2 from file1.
```

(back to top)

Acknowledgments

A list of resources which are helpful and would like to give credit to: * Flask kerberos * Python kerberos module