L-Università ta' Malta
Faculty of Information & Communication Technology

Project Report

Manwel Bugeja

February 17, 2021

Contents

1 Question 1

1.1 High level architecture overview

1.1.1 Connection

A Unix domain socket server and client where created in python. TLS wrappers from the SSL library were used to enable communications over a secure channel. Socket creation and wrapping is shown in listing 1

The information to be communicated is stored in a python dictionary. Before being sent, the dictionary is dumped into a string which is then transferred.

Listing 1: Socket creation and TLS wrapping

```
client = create_connection((ip, port))
tls = context.wrap_socket(client, server_hostname=
   hostname)
```

1.1.2 Server

The server has two functions to handle the two types of applications, these being *handle_application()* and *handle_client()*. The server receives messages from the client as a json string of bytes but then transforms it to a python dictionary to be easier to parse.

Using if statements the admin/application handler function does the required operation and sends the input back to the client.

1.1.3 Client

The are two types of clients to make the code simpler. An application client and an admin client. They both start off with initializing the connection to the server and proceed to go through an endless while loop for continuous communication.

1.1.4 Keys

Key operations are handled by a file called *Keys.py*. This file includes key generation, key derivation and key verification.

When an admin tries to log in, he is prompted for a username and a password. Using these credentials, the admin's part of the Master Key (MK) is generated. The password is used as a password for the key derivation while the username is used as a salt. This is done to prevent dictionary attacks.

The *PBKDF2* key derivation function was used. SHA256 was used as a hash module but could have easily been substituted with SHA512 for added security.

The Key Encryption Key (KEK) is derived from MK1. This process is also used for admin verification since the MK in never stored in memory. MK verification is done as follows:

1. Admin1 connects and his part of the MK is generated
2. Admin1 waits until the second admin connects and is verified
3. When Admin2 connects, his part of the MK is generted
4. A new KEK is generated from MK1 (which is derived from the newly generated MK)
5. The new KEK is compared to the original
6. If it is the same than both admins are the real ones
7. If it is incorrect than one of the admins used an incorrect user/password
8. Admin2 is kicked out of the system* and can try to login again

*This was the design chosen for code simplicity. If this system was going to be deployed, this design choice would never be taken. A malicious intruder can login as admin one and deny the real admins from using the system since credentials can not be checked without both admins being logged in. One way to deal with this would be to use a flag so that while admin 1 is waiting to be verified, the while loop checks on the flag. Failure to verify would result in the flag being set to false result in both admins being kicked from login. Another way would be to introduce a timeout where an admin is forced to log out after x amount of seconds of waiting to be verified.

2 Question 2

3 Question 3

4 Question 4

First and foremost, the MK should never be stored in memory. In all of its uses, the MK should be derived from both MK parts which means both

admins need to be logged in. Passing the MK as a parameter should be used as a last resort as this creates a copy of the MK in memory and destroys it when the function in question exits.

Unused instances of MK parts should be destroyed (set to null) to protect the MK. This also applies to keys derived from the MK although..

References