universidade
de aveiro

Project 2 – Authentication, equipa 18

Domains:

- uap

  127.0.0.1:4000

- app_auth

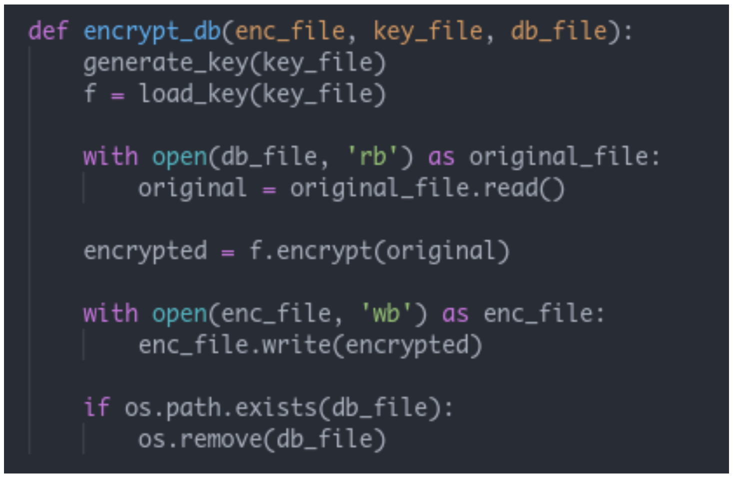
  127.0.0.1:8080

Database Encryption

For this version of the project, we improved our database's security by encrypting them as a whole. The key to decrypt the database changes every time it gets used.

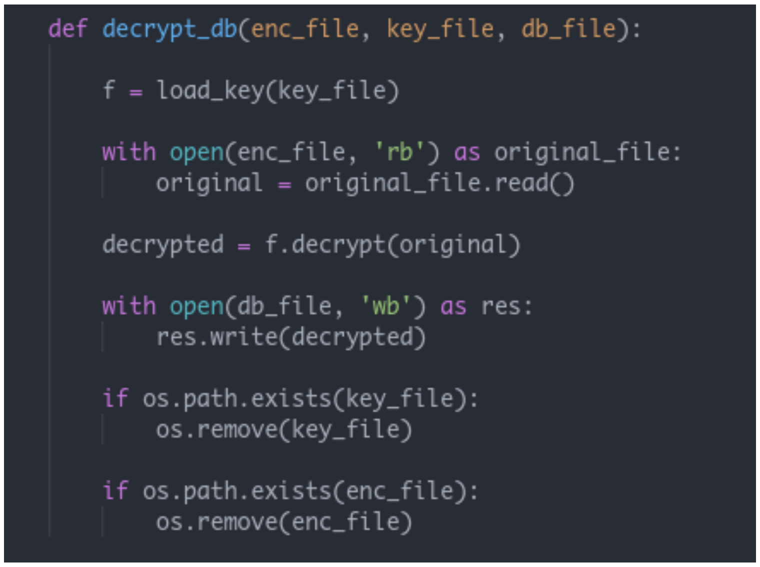
This implementation can be found in the file access_db.py found in both the uap and app_auth applications.

Encryption:



```
def encrypt_db(enc_file, key_file, db_file):
    generate_key(key_file)
    f = load_key(key_file)

    with open(db_file, 'rb') as original_file:
        original = original_file.read()

    encrypted = f.encrypt(original)

    with open(enc_file, 'wb') as enc_file:
        enc_file.write(encrypted)

    if os.path.exists(db_file):
        os.remove(db_file)
```

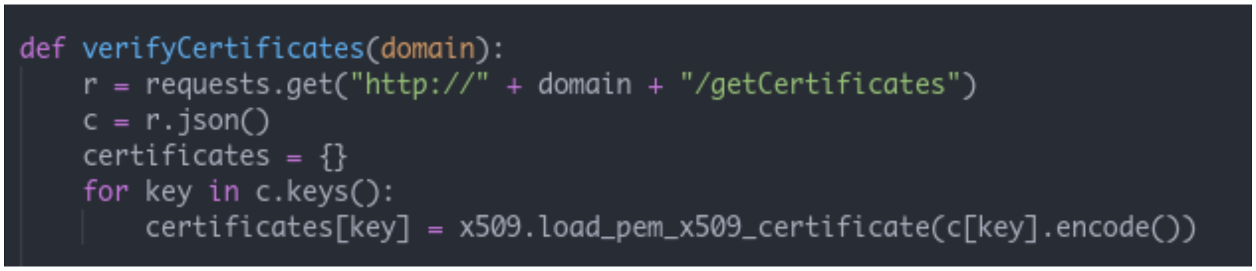
Decryption:



```
def decrypt_db(enc_file, key_file, db_file):

    f = load_key(key_file)

    with open(enc_file, 'rb') as original_file:
        original = original_file.read()

    decrypted = f.decrypt(original)

    with open(db_file, 'wb') as res:
        res.write(decrypted)

    if os.path.exists(key_file):
        os.remove(key_file)

    if os.path.exists(enc_file):
        os.remove(enc_file)
```

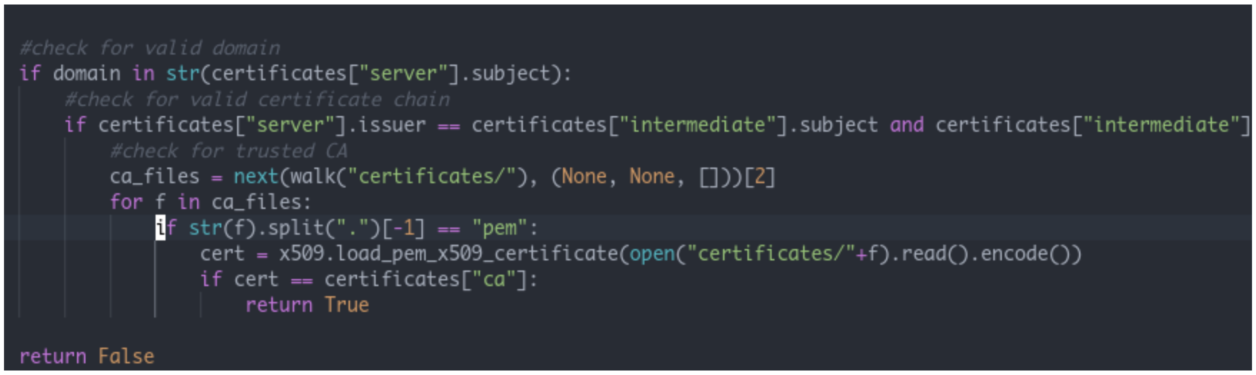
Certificates

In order for the uap to verify it is communicating with a trusted server, it asks the server's api for its certificate chain:



```
def verifyCertificates(domain):
    r = requests.get("http://" + domain + "/getCertificates")
    c = r.json()
    certificates = {}
    for key in c.keys():
        certificates[key] = x509.load_pem_x509_certificate(c[key].encode())
```

After that it makes the necessary checks for the certificate's validity, and check if it is signed by a trusted certificate:



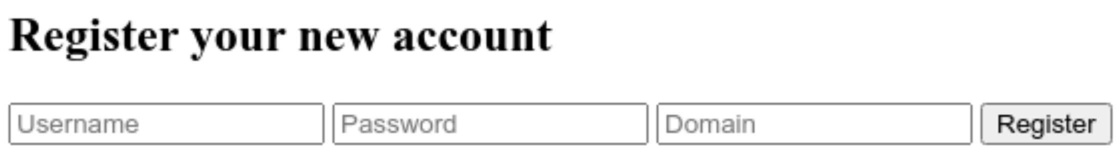
```
#check for valid domain
if domain in str(certificates["server"].subject):
    #check for valid certificate chain
    if certificates["server"].issuer == certificates["intermediate"].subject and certificates["intermediate"]
        #check for trusted CA
        ca_files = next(walk("certificates/"), (None, None, []))[2]
        for f in ca_files:
            if str(f).split(".")[-1] == "pem":
                cert = x509.load_pem_x509_certificate(open("certificates/"+f).read().encode())
                if cert == certificates["ca"]:
                    return True

return False
```

In order to generate the certificates, we used oppenssl. The openssl configuration used to generate the root CA certificate can be found in root_ca/openssl.conf.
The intermediate is signed by the root CA and the server certificate is signed by the intermediate.

UAP - Front End

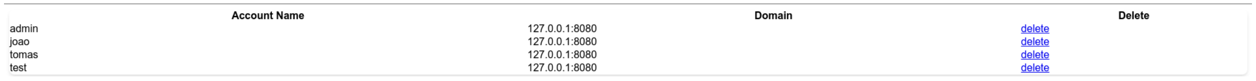
“/registerUser” :

Adds account to uap database



“/manageUsers”:

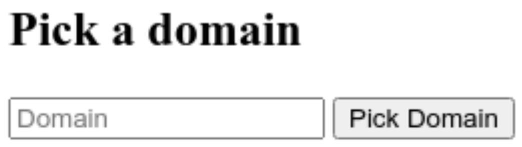
Allows the user to see all accounts and delete them if needed.



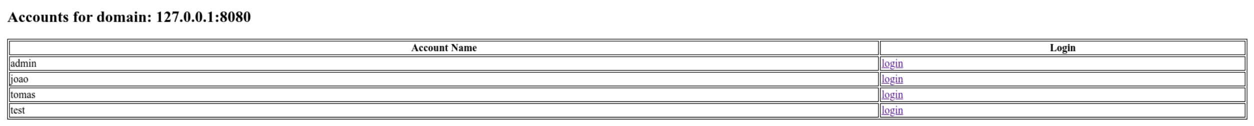
Account Name	Domain	Delete
admin	127.0.0.1:8080	delete
joao	127.0.0.1:8080	delete
tomas	127.0.0.1:8080	delete
test	127.0.0.1:8080	delete

“/domainPicker” :

Allows the user to pick a domain to login.



Leads to “/login”
And shows known accounts for the chosen domain.



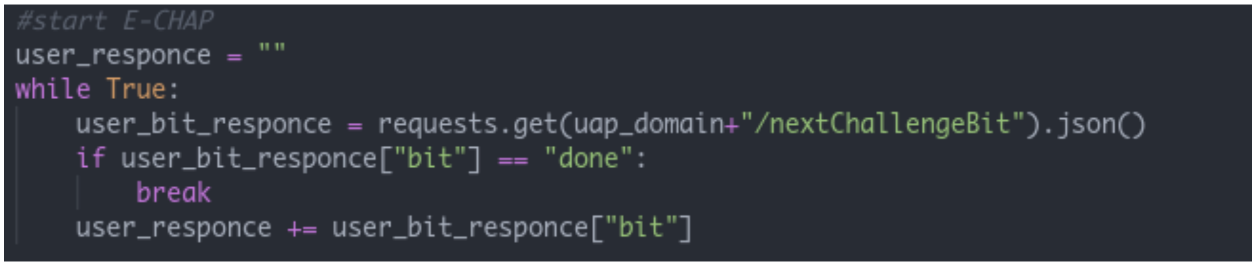
Accounts for domain: 127.0.0.1:8080

Account Name	Login
admin	login
joao	login
tomas	login
test	login

Used app_auth’s domain for exemple.

E-CHAP

After the user selects an account to login with the uap sends a request to the server to start the authentication process.
Then the server responds back with a nonce. Both the uap and the app_auth calculate a challenge answer (md5(password + nonce)).
Then the server requests the uap’s api for the response, bit by bit.



```python
#start E-CHAP
user_responce = ""
while True:
    user_bit_responce = requests.get(uap_domain+"/nextChallengeBit").json()
    if user_bit_responce["bit"] == "done":
        break
    user_responce += user_bit_responce["bit"]
```

The server only stops requesting when the uap is done, so a hacker would have to guess both the contents and the size of the responce.