End-to-End Secure Chat

Brogrammers: Tyren Villanueva & Huy Le

Main Project:

- To create a secure message application between two clients.
- Establish a HTTPS server
- Encapsulation and decapsulation on client
- A+ SSL certificate

Resources:

Server Side:

- AWS EC2 instance
- LAMP server
- Let's Encrypt and Cerbot
- Tried to use Node.js and MongoDB

Client Side:

- Python 3.6.1
- Cryptography library (https://cryptography.io/en/latest/)

Server Side Development:

- Setup a LAMP server on AWS EC2 instance Ubuntu
- Used Cerbot to get a certificate from Let's Encrypt and enable HTTPS
- Got A+ grade on SSL Labs
- Decided to work with Node.js and MongoDB
    - Developed locally then deploy to server side when completed

Client Side Development:

Encryption:

- Using AES 256-bit key and 128-bit IV.
- Generate integrity tag with HMAC SHA256.
- Encrypt the keys with RSA public key.

Decryption:

- Decrypt the keys using RSA private key.
- Use HMAC SHA256 to re-create the tag.
- Compare the tag (return failure if not match) then decrypt the message using AES.

Implementation:Detailed

Websocket was chosen as messaging protocol.

Message Encryption-

-open public key pem file and import to RSA- `rsa_public = serialization.load_pem_public_key(f.read(), backend=backend)`

-generate an Initialization Vector and AES key

-create a cipher object by combining AES algorithm with CBC mode..... Used to create AES object `aes_object = cipher.encryptor()`

-create a 256 bit key used for HMAC

-HMAC object is created using HMAC key and SHA 256- `hmac_object = hmac.HMAC(hmac_key, hashes.SHA256(),`

-create integrity tag `tag = hmac_object.finalize()`

- concatenate the aes and hmac keys

Implementation:Detailed (cont...)

```
rsa_cipher = rsa_public.encrypt(
        concatenated_key,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
```

-Encrypt the concatenated keys and apply optimal asymmetric encryption padding.

-Create and return a JSON output

Implementation:Detailed (cont...)

Message Decryption-

-Open and import the private key to an RSA Object

-Decrypt the rsa cipher concatenated keys

```
concatenated_key = rsa_private.decrypt(
        json_object['RSA_cipher'],
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
```

-Retrieve the aes and hmac keys separately by splitting the concatenated key...

```
aes_key = concatenated_key[:len(concatenated_key)//2]
    hmac_key = concatenated_key[len(concatenated_key)//2:]
```

Implementation:Detailed (cont...)

-Recreate tag to compare with original tag to ensure integrity.

-Decrypt the message to plaintext after tag is proven valid

Authentication Method

We used Json Web Tokens for authorization.

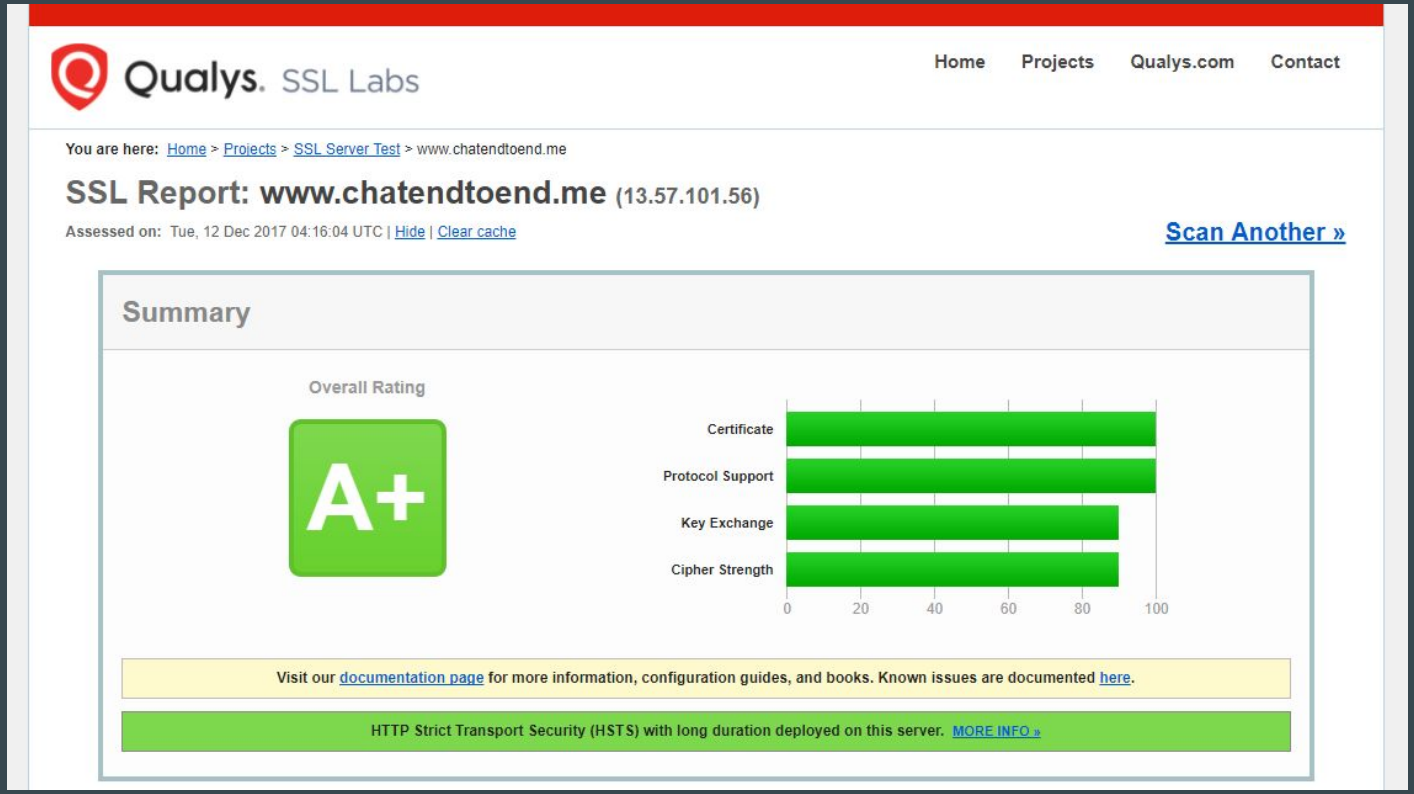
We used Passport middleware to authenticate requests.

All API routes are authenticated by username and password..... Token is passed back upon successful login

Key Exchange

Users that would like to communicate through our application will send their public keys through email.

SSL Labs Grade:



Questions: