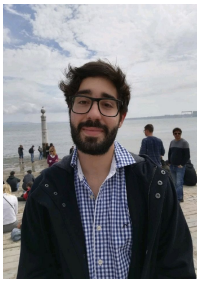
Smartphone as a security token

Network and Computer Security

Group A31

Tomás Silva
83862



Ricardo Gueifão
87699

Samuel Vicente
87704

1. Problem

The chosen topic is regarding a smartphone as a security token. In this scenario, a bank has one server which hosts one web app. When accessing the web application a user needs to perform two-factor authentication using our mobile application to prove their identity. Another authentication is needed when the user attempts to perform a critical operation.

Due to the sensitive information and operations that a bank provides, the main security problem that needs to be solved is to protect the access to an account.

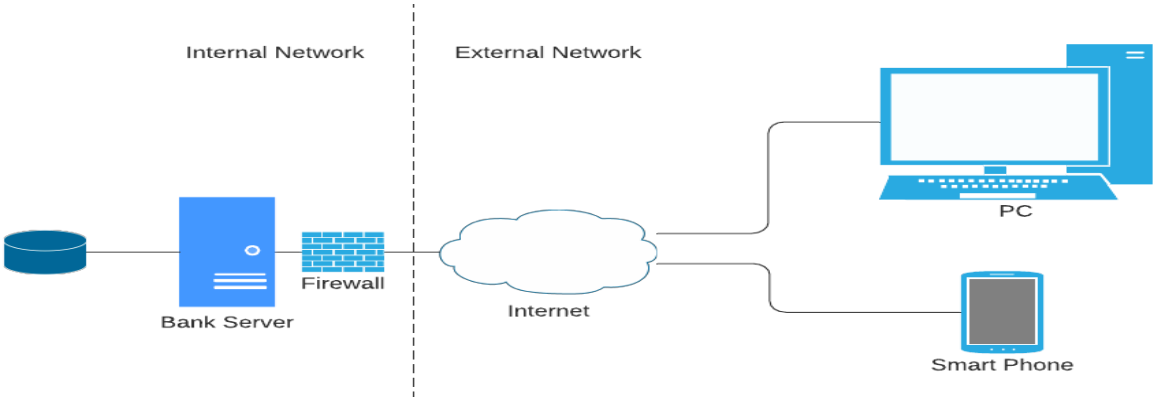
1.1. Requirements

- **Confidentiality:**
  - The data passed between the server and the client (both in the web application and in the smartphone) must be encrypted
- **Integrity:**
  - The data passed between the server and the client (both in the web application and in the smartphone) can not be tampered with and should be preserved.
- **Authentication:**
  - To connect to the server of the bank the user must perform two-factor authentication
- **Non-repudiation:**
  - The critical data or operations passed between the server and the client (both in the web application and in the smartphone) must be signed
- **Freshness:**
  - The data passed between the server and the client (both in the web application and in the smartphone) must have an incremental nonce.

1.2. Trust assumptions

- Fully trusted:
  - Server
- Partially trusted:
  - Client Laptop
  - Client Smartphone

2. Proposed Solution



2.1. Overview

Our solution consists of three elements, a server that handles the bank operations (check account information, deposit and transfer) and a client that connects to our server via a web application that wants to perform these operations and the client smartphone that acts as a security token.

In order to login to our web application, the client needs to perform two-factor authentication (something you know: the password to log in and something you are: the biometric fingerprint that the smartphone reads). When authenticated the user gains access to the account and it can perform different operations. We divide these operations into two categories: critical and non-critical. The non-critical operations are read-only operations (check account information and transfers), the critical operations are the ones that change the account (deposit and transfers), to perform these operations the user must grant authorization (using biometric fingerprint in the smartphone linked to your account) using the smartphone.

2.2 Deployment

The server is deployed on a Linux machine hosted at digitalocean.com and our web application is currently accessible through https://sirsbank.tk/bank/. Our server has a valid certificate signed by Let's Encrypt and our web application is only accessible using HTTPS. The server was configured using Apache 2.

2.3. Secure channel(s) to configure

The bank server has 2 certificates, one signed by Let's Encrypt to perform HTTPS communication with the web app and another signed by our CA to that secures the communications with our mobile application. The CA's root certificate comes preloaded in the client's mobile application.

2.4. Secure protocols to develop

Properties

Our custom protocol offers confidentiality when it matters for instance when transporting the token that registers the phone to an account, everything else is open since it doesn't matter, for instance, the public keys of the client and the certificate of the server. Freshness is achieved with the use of nonces. Integrity and non-repudiation are achieved with digital signatures.

We don’t need to transport any information since the fact that the user is able to establish the connection means the user is in possession of the protected keys and so is authenticated.

Languages used

Our server was made in python using Django to serve the web app. Our mobile application was made in Java.

Key Distribution

Our server starts with a pair of RSA2048 keys generated, a certificate signed by Let’s Encrypt and its private key and a certificate signed by our CA and its private key. The first certificate is used to provide HTTPS access to the web application, the second one to secure the communication channel between the server and the mobile app. The client, that comes preloaded with our CA’s root certificate to verify the server’s certificate, then generates its own RSA2048 key pair, protected by biometrics, to sign its messages, this RSA key pair is stored using Android KeyStore meaning only this app can access it, the keys can never be exported, only used and it’s protected with the user biometrics.

3. Results

In our final solution, we developed two different custom protocols and used HTTPS to secure the messages exchanged in the secure access via the web. In Annex 1 we can see the two custom protocols used. The ‘register’ protocol is used when a client registers to the web application and the ‘login’ protocol is used when a client uses the smartphone to grant authorization to perform an operation and to log in. We came to the conclusion that there was no need to to use AES and Diffie-Helman as we only need to send a message that confirms the identity of our clients. This is done in ‘register’ when the client sends the token along with the client’s public key to bind the public key to an account and it is done in ‘login’ where the server verifies a message that is signed with the client private key, using the public key that was

bound to the account in ‘register’.
We also protect our server communications with the web browser using HTTPS and our machine has a firewall (in Annex 2) that prevents DDOS by dropping packets with TCP packets that don’t make sense (for example all flags set, or with no flags set), and ICMP packets to prevent ping-of-death attacks.
To conclude our solution provides us with the following:

Authentication

The Authentication is ensured by the validation of the server certificate in the mobile app, and the server verified that the public key that received is the key of the user that logged in.

In the case of the registration, the User is presented with a challenge (send the token displayed in the signup page), if the challenge is completed successfully the server links the user with his public key.

Perfect Forward Secrecy
Our protocol does not assure perfect forward secrecy, however it doesn’t need to. Since the client only needs to send one message to the server to prove it’s the identity we can get this with the message that sends the token (in the registration protocol) and with the message signed by the private key (in the login protocol). We considered Diffie-Helman to secure the channel, however, this messages needed to be exchanged during the handshake which would render any Diffie-Helman implementation worthless, we would secure the channel with AES and then immediately close it.

Confidentiality
Confidentiality in the communications with the web application is assured by the use of HTTPS, in the mobile application communications we protect the token in the registration protocol by encrypting with the server’s public key which assures us that only the server can read this token. This token is critical information as it is the base for authentication since it binds one public key to one account.

Integrity
We can assure data integrity on communications with the web app by using HTTPS, and because of that the data cannot be modified or corrupted during transfer. We assure data integrity on communications with the mobile application with the use of digital signatures which show that the data has not tampered.

Freshness
Finally, during the communication between the mobile app and the server, the server

creates a nonce, afterwards, the mobile app responds with nonce + 1, so replay attacks are mitigated.

Non-repudiation

We offer non-repudiation in the messages between the server and the mobile application by the use of digital signing.

Firewall

We set up a few rules in iptables to set up our firewall. The aim of the firewall is to mitigate DDoS attacks and for that we block invalid packets with rule 1. rule 2 to 8 block TCP packets that have flags that don't make sense (for example rule 2 blocks TCP packets with flag SYN and FIN set up. This doesn't make sense since the SYN flag should only be set up when starting a transmission and the FIN flag when ending one. There is no plausible scenario in which both flags should be up and so we block the packet). Rule 9 block ICMP packets since the protocol will not be used and an attacker could try to perform a ping-of-death attack.

4. References

1. OpenSSL
2. Java
3. Python
4. SQLite
5. Apache2
6. Python cryptography
7. Google Biometrics
8. Django
9. Let's Encrypt

Annex 1:

Register:

Login:

Annex 2:

```
#!/bin/bash

#Block invalid packets
iptables -t mangle -A PREROUTING -m conntrack --ctstate INVALID -j
DROP
#Block packets with nonsens TCP flags
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,SYN
FIN,SYN -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags SYN,RST
SYN,RST -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,RST
FIN,RST -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,ACK FIN -j
DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,URG URG -j
DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,PSH PSH -j
DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL NONE -j
DROP
#Block ICMP packets
iptables -t mangle -A PREROUTING -p icmp -j DROP
```