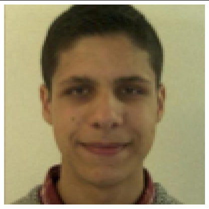
TÉCNICO LISBOA

Network and Computer Security

Masters in Information Systems and Computer Engineering

Alameda - 2018/19

Group 27

		
Filipe Martins	Vasco Guita	Ziran Wei
83458	83572	92261

Remote Document Access

Problem

We’re aiming to develop a remote file accessing software that allows users to store, view, share and modify their remotely stored files. Since this will be a cloud-based solution over a public network we need to address several issues related to security, privacy and authenticity. We need to make sure that the files aren’t viewed or modified by unauthorized users and that their contents are encrypted so as to not allow attackers to capture them while they’re being sent over the network or even if they gain physical access to them. There are several other attacks such as ransomware attacks that also need to be prevented. By trying to solve all these problems we’re providing the users a way of accessing their files without having to worry about their physical location (as long as you have our software installed and a working internet connection you should be able to access them).

Security Requirements

1. A user needs to be authenticated in order to view, modify and delete their files.
2. Confidentiality on the remotely stored files : This is necessary because we don’t want unauthenticated users to have access to the contents of the files even if they get physical access to the hardware.
3. An attacker that tries to sniff out the packets of connections between the client and the server cannot figure out their contents or tamper with them. Therefore, we must ensure we have confidentiality and integrity over network communications.
4. The client must verify the authenticity of the server in order to prevent man in the middle attacks. This will prevent users to send their credentials to attackers.
5. Freshness of the packets sent over the network to prevent an attacker from repeating data transmissions and possibly overwriting files in the server.
6. The authentication should be resistant to brute force attacks. This will prevent attackers from trying to guess a users’ password and gain access to their files.
7. Confidentiality of the passwords stored in the server. This is to make sure that even if an attacker has physical access to the server he is unable to get the users’ credentials.
8. Integrity in the remotely stored files. This is a requirement because we want to prevent undesired changes to the files. These changes include ransomware attacks or unauthorised modifications.
9. Non-repudiation : In the network we must guarantee that the authenticated user is the one who sent the message to the server. In the server we must track each change to the files and bind them with its author.

Proposed Solution

1. We will have a login system that requires a username and a password from a user which in turn will be sent to the server whenever he/she wants to access its files. The server will then verify the credentials and make sure the user has permissions.
2. The files stored in the server will be encrypted using AES-256 with a DEK (Data Encryption Key) randomly generated in the client. The very DEK is encrypted with a KEK (Key Encryption Key) which is the users public key. This will allow the authenticated user to download the encrypted file and the encrypted DEK, decrypt the DEK with his private key and with the DEK decrypt the file. All this in the client side preventing attackers to gain access to its contents (to do this we will use a PGP implementation).
3. To ensure confidentiality, freshness and integrity of the transmissions between the client and the server we will use HTTP over TLS for these connections. This protocol will generate a short-term session key from the long-term public and private keys to encrypt the packets containing the files that are transmitted through the connection established between the client and the server.
4. HTTPS over TLS also solves this problem because it requires a public key certificate from the server in order to verify its authenticity.
5. In order to resist brute-force attacks we will establish a 3-second delay between each login attempt to the same account. This will lower the probability for an attacker to find out the password in a viable time.
6. In order to grant confidentiality of the passwords in the server we will have a table in a database with the username of each user and it's password's hash. We will use SHA-3 to generate 256 bit hash for each password. When the user authenticates the password sent to the server will be hashed and this hashed compared to the one stored in the database. This way we never store a password in plain text in the server.
7. We will implement a backup system that helps us to recover from any undesirable changes to the files, thus protecting from ransomware attacks as well.
8. To ensure non-repudiation on the messages sent over the network by the client to the server will be digitally signed (using the client's private key). The messages sent by the server to the client are protected from repudiation by the http over TLS protocol, because the server digitally signs each message. On the server we ensure non-repudiation by keeping a log of every change made by the users to the files.

Results

We managed to implement the following proposed solutions :

1. For the authentication with the server we used HTTP Basic Access Authentication. To authenticate all the requests in the PKD we use a basic password hash comparison.
2. This was implemented like originally planned using the BouncyCastle library to have access to openPGP actions.
3. We obtained a A NS record of the www.sirsrda.tk and ca.sirsrda.tk to a public server with the ip 193.136.164.218 and then we obtain a Let’s Encrypt certificate using the certbot tool. And then we redirected all the traffic on the port 80 to the port 443 of the server using nginx.
4. Solved by 3.
5. We only managed to implement the protection against brute-force attacks in the connections between client and PKD (10 seconds after each failed login attempt). In the server side the brute-force protection is included in the BCrypt implementation.
6. Instead of SHA-3 hashes like we originally planned we used the Spring Security with BCrypt instead due to ease of implementation since we were already using that library.
7. The ransomware protection would be implemented using server replication to at least reduce the risk.
8. Non-repudiation was also implemented like we originally planned to. We didn’t implement the log for the file modifications.

Extra : We had all the tools to implement the file sharing mechanism but we didn’t manage to do it on time. Also the file signing mechanism with shared files would have to need a different approach which would be keeping track of the last person who modified the file in order for the next client who gets the file know which public key he needs to obtain in order to verify the signature.

Evaluation

Since we only store encrypted files in the server an attacker who has physical access to the machine won’t be able to figure out its content because we use PGP to encrypt the key used to encrypt the file. Using that we would also provide a way for clients to share their files through the server without it ever having access to the decrypted content.

Though there is a weakness in our proposed solution: If an attacker has access to both the main server and the backup server he may modify (executing a ransomware attack for example) the encrypted file and disable the recovering of the same.

Conclusion

Unfortunately we did not manage to fully implement what we aimed to in our initial proposal, but we managed to implement the core of the project, to the point where signature verification with our public key directory(which we call CA) would be enough to grant integrity. The project is ready to incorporate sharing functionality through the sharing of the data encryption key encrypted with the key encryption key of the receiver(PGP public key).

Tool References

- Mariadb : To store the user ids and the hashes of their passwords as well as the files relations they have with the users
- certbot: to install letsencrypt certificates to use HTTPS over TLS in the communications between the clients and the servers
- Java Version 8 with JCA framework: To implement the client and the server modules.
- OpenKeychain API: OpenPGP Library for Java which will be used for encrypting, and decrypting files.
- Spring Boot (Along with Spring Web, Spring Security)