Secure Scheduler

Project Report

Tristão da Câmara (67092), Guilherme Duarte (67601) and António Machado (68122)

Group 40

1 Introduction

The secure scheduler application allows each user to create their own schedules that are saved on a local device. The application also allows users to arrange appointments with each other by analyzing the other users' schedules and the respective free time slots. The meeting may then occur if both users have compatible free time slots. Furthermore every user is able to accept or reject a meeting request, as well as cancel or reschedule an appointment already in the schedule. An external server exists in order to establish a secure communication between the users.

2 Motivation

Generally speaking most people are aware that they have information in their life that is worth protecting. But do those people recognize the bits and pieces that go into understanding what is confidential? So, lets take the example of strategy of a firm targueting another for a hostile takeover. What is the information that provides pieces to the puzzle that would allow an outsider to know what the acquirer's intentions are? It is not just the acquirer's internal documentation or even it's current available funds, it is also a meeting between the acquirer's CEO and the CEO of one of the shareholders. This is where the secure scheduler comes into play, protecting our personal calendar is another step towards the assurance of our privacy in the modern digital era.

3 Goals

3.1 Basic

- Users must be able to schedule meetings;
- Each user must be able to keep his/her schedule saved on his/her local device;
- Each user may accept/reject a meeting request by another user;
- Each user may cancel or reschedule an existing meeting.

3.2 Intermediate

- Establish a secure communication channel between a user and the third party server in order to prevent information leakage;
- Establish a secure communication channel between users in order to prevent information leakage;
- Assure that the application is able to find a meeting between two users, without revealing eachother, their personal schedule.

3.3 Advanced

- Assure that the application is able to find a meeting between multiple users, without revealing eachother, their personal schedule;
- Assure that the shared keys (KEK) are securely stored both in the server and clients' sides.

4 Assumptions

- The calendar assumes only one civil year, in this case 2014.
- The password database in the server is secure.
- The server and each user share a secret key that was previously generated and distributed using Diffie-Hellman, in order to authenticate both parties;
- The application is prepared for only two clients which have static functions. Alice only performs meeting requests and Bob only responds to them.

5 Application Design

Please see Fig.1 on the appendix.

6 STRIDE Analysis

6.1 User Login

1. A malicious user accesses or tampers the schedule information saved on the local device (Tampering of data);
2. A malicious user may perform a dictionary attack in order to extract the password of the legitimate user (Information disclosure).

6.2 Server-User Communication

1. A malicious user views or tampers with personal data en route from the to server to the client or from the client to the server (Tampering with data/Information disclosure);
2. An attacker denies access to the user database server computer by flooding it with connection requests (DoS).

6.3 User-User Communication

1. A malicious user views or tampers with personal data en route from one client to the other (Tampering with data/Information disclosure);
2. An attacker deletes or modifies the dates of the meeting requests (Tampering with data/Repudiation);
3. A malicious user impersonates a legitimate user in order to gather confidential data (Spoofing identity/Information Disclosure).

6.4 Application-Level

1. A malicious user may infer other users' schedule availability through extensive analysis of multiple meeting requests (Information disclosure).

7 Elimination or Mitigation of the Vulnerabilities

7.1 User Login

The user's personal information is stored on the local device. This information is only accessible if the user knows the password for the application The password itself is not stored locally, instead the device stores the hash of the password which it will compare after each login attempt. The *hash* is generated by appending a cryptographic *salt* to the password and going through 5000 iterations of the *PBKDF2* function. This number of iterations provides a good balance between increased security and the inconvenience of longer pauses when logging in to an account. The hash is then verified by a *slowEquals* function that also extends the time for the comparison process. This way the application assures the decrease of efficiency of a bruteforce and/or dictionary attack, without affecting the user experience.

7.2 Server-User Communication

All the communications between the server and the users are assured through encrypted messages. The message encryption uses the AES algorithm along with a 256bit key which is the recommended size nowadays for communications that want to meet modern security requirements. As stated above, this key should be generated through the *Diffie-Hellman (DH)* Key Exchange protocol, which we will describe thoroughly in the next subsection. By using *DH* we prevent eavesdroppers from reading the messages exchanged between the users and the server. We made the assumption that the files were safely stored both in the server and in the clients' devices. We could, however, have encrypted the shared keys with the user password on the clients' side and then decrypt them using the password with the Pretty Good Privacy protocol, which would add an extra security layer to our application. Moreover if an attacker tries to tamper with the data, the recipient will know it happened

since the decrypted message becomes unreadable. The user will then dismiss it and act accordingly. In order to prevent DoS attacks the server runs in multiple threads, if no data is sent through a connection for more than 30 seconds it gets closed. To further prevent the server from DoS attacks a *Packet Filter* firewall type should also be installed in the server.

7.3 User-User Communication

The communication between users is again performed by using the AES encryption algorithm with 256bit keys. The most suitable approach is to generate a session key each time a user wants to communicate with another; the server generates the sessions keys and distributes them through messages encrypted with a KEK (Key Encryption Key). However this approach by itself does not provide *Perfect Forward Secrecy*. One way of tackling this problem the implementaation of the Diffie-Hellman key exchange protocol. This protocol does not share the session key through the communication channels instead both ends agree on two public values. After setting the public values each participant chooses a private value, which together with the public value will generate a public key, that may be exchanged through insecure communication channels.In the next step both users generate the same session key using their private keys and the public keys previously exchanged. However, since the public values are sent through insecure channels this protocol is susceptible to both *Man-in-the-Middle (MITM)* and *Replay* attacks. We thought of a solution that prevents the MITM and the Replay attacks (see Appendices) that we were not able to implement in our project: 1) The two initial public values are generated by the server which then sends them to both users through messages encrypted with their corresponding KEKs; 2) The public keys generated by the users are sent again to the server, which forwards them to the correct users;3) Once the users receive their public key and generate the session key, they challenge each other directly. The challenge consists of messages containing a cryptographic nonce and a timestamp encrypted with the session key. This protocol prevents both a MITM attack since the attacker cannot eavesdrop on the conversation while spoofing his identification, and a Replay attack since the nonce prevents a message from being used twice. The timestamp stipulates for how long an used nonce must be kept in storage, discarding the need of strict clock synchronization between all parties.

7.4 Application-Level

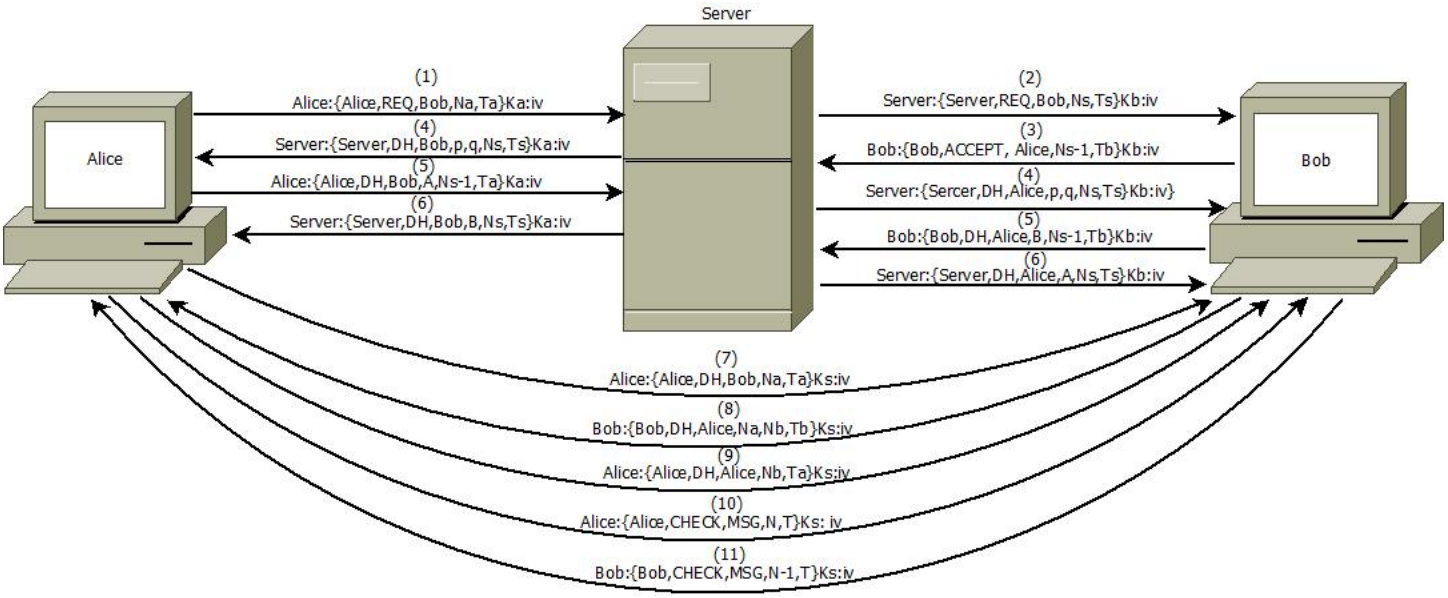
Although we do not have any measure(s) implemented to eliminate the threat posed by the analysis of the meeting requests, we would like to discuss some of the existing possibilities: 1) Sorting through a calendar in a specific order allows the analysis of patterns in the searches therefore one possible solution would be to randomize the order in which the calendar is traversed, always taking into account the date limits established for the scheduling. 2)If a scheduling request that goes through the whole calendar of a user returns no results, both users end up infering each others calendars based on their personal ones. A countermeasure would be to, together with the random searching, limiting the number of queries for free slots that the application can produce per request. This way it hands over the control of the situation to the queried user, allowing him to reject further requests.

8 Conclusion

The functionalities that the developed application posesses are only to be viewed as a platform for the development of the required security measures. We aimed at developing secure communications for each type of exchanged message. We successfully cyphered the messages exchanged by the users and server as well as implemented the Diffie-Helman protocol which could not be properly tested, due to unrelated problems with parsing of the cyphered messages. We believe that that although there our implementation has flaws we could not mitigate, the security analysis performed is according to the expectatons of such a project. We would also like to highlight that the implementation flaws are not produced by flawed protocols but instead by specific implementation problems such as the parsing of the exchanged messages.

Appendices

A Figure 1



References

[1] Network Security Essentials, William Stallings, 2003.

[2] Computer Networking, A Top-Down Approach Featuring the Internet, James F. Kurose, Keith W. Ross, 2005

[3] Link for the client–server communication implementation: `http://beej.us/guide/bgnet/output/html/multipageclientserver.html`

[4] General coding doubts: `http://http://pt.stackoverflow.com/`

[5] PBKDF2: `http://en.wikipedia.org/wiki/PBKDF2`

[6] PGP: `http://www.pgpi.org/doc/pgpintro/`

[7] Password hash: `https://crackstation.net/hashing-security.htm#javasourcecode`

[8] Symmetric cryptography: `http://www.javamex.com/tutorials/cryptography/symmetric.shtml`