Xwriter 3000

Remote document access project
For the course
Network and Computer Security

Group 33
Alameda

- Diogo Peres Castilho - 78233
  diogo@castilho.me
- Arthur Vernet - 88895
  arthur.vernet@epfl.ch
- José Augusto De Góis Rodrigues de Sá – 90854
  joseagrsa@gmail.com

Problem

Xwriter 3000 is a cloud application that allows people to write books together, and get feedback on their writing by the people they select (friends, family, correctors, editors, …) by sharing their work.

The main issue that arises from this description is how will we ensure the confidentiality of the authors' work. This was the main focus during the development of the application as we want authors to have trust in our platform, they must be assured that their work won't be stolen, copied or even looked at by unwanted people (including us) before they decide to publish them.
As we described it, the authors have the possibility to pick the people that are authorized to see their work and/or contribute to them. Which leads us to the second biggest issue (that was also a priority during the development), identify and authenticate the people who have the right to access a given book while preserving the confidentiality and integrity of it.
Since the application is running in the cloud, and therefore our server contains all sensitive data, we had to ensure the data is unreadable to anyone not proprietary of it (including us, once again) in case someone was to take control of the server.
Finally, we want authors to be able to work on their books wherever they are and whenever they want hence the decision to propose a robust server against denial of service attacks and unpredictable accidents such as infrastructural failures by providing a recovery server, but also the possibility to access their work on any machines.

Requirements

The following security requirements were identified to achieve the goals we set:

- Privacy of the work for a safe storage
- Identification and authentication of the authors
- Secure channels between the authors and the server
- Safe sharing of the work between authors through encryption and identification
- Recovery mechanisms to guarantee integrity and availability of the books

Solution

Let's first discuss the basic solution and then empirically go from there to the intermediate and then advanced solutions.

Basic solution

We decided the basic solution was to be a simple version of the application we are proposing. It doesn't include any security feature, but all the core functionalities meaning everything from creating an account, logging-in, creating a book, selecting additional authors and/or readers, and writing it was possible.

The log-in mechanism includes session IDs which are big random String generated when the user logs in and that is checked every time the client communicates with the server.

In addition of all that, to simplify testing a graphic and interactive user interface is included.

Intermediate solution
The intermediate solution allows authors to revoke authorizations they gave and also includes everything related to encryption: encryption of the communication and encryption of the data stored.

A pair of keys is generated whenever a client creates an account. Along this pair is generated a 16 bytes code which allows the user to log-in on different machines. This code is generated locally and is never shared with the server. It is used in addition with the user password to cipher his private key so that it can be stored with his public key on the server. This way when the user tries to connect on a different machine, the client can request the same key pair from the server, decipher the private key with the code and the user password, and use it again to authenticate himself to the server. The goal of this 16 bytes code is to allow the storage on the server of the user key pair in a secure fashion, so that only him can decipher it and therefore reuse it if he changes of machine.

Whenever a client creates a book, a symmetric key is generated with which the book is always ciphered before being sent to the server. The client then ciphers the symmetric key with his password and his 16 bytes personal code and sends it to the server to store it. This way, if the user changes of machine he can, as explained above, access and read his book by requesting his key pair and then the symmetric key.

Whenever a client wishes to share a book, he asks for the symmetric key of the book and the public key of the user he wants to share it with. He then deciphers the symmetric key with his password and his 16 bytes personal code and re-ciphers it with the public key of the chosen user before sending it to the server. The chosen user can now at any time request this symmetric key and the ciphered book, decipher the symmetric key with his private one and decipher the book to work on it.

Therefore, the private keys, the book symmetric keys and the book text is always stored on the server safely. These are all sensitive data, stored on a remote location, but only readable by those who are allowed to.

Now, the communication between clients and server are done always the same way. The message is ciphered with the receiver public key (the client knows beforehand the public key of the server, and the server stores the client public key when he creates the account) and signed (with a digest of the message and the sender public key). This ensures confidentiality, integrity and non-repudiation of the message.

The author creation works as follows. The client sends the password, the username and the key to generate the HMAC. This first message is not signed but is ciphered with the server public key. Then the user sends his private key (ciphered with his password and 16 bytes personal code) and public key with an HMAC generated by the same key used before.

The log-in request is a message containing the password and the username, it is ciphered with the server public key but not signed. The answer of the server is however signed and ciphered with the client public key.

Advanced solution
The advanced solution includes a recovery server which takes over the main server if this one does not give any life proof after a certain time. It also includes a basic firewall, which analyses if the messages he receives make sense for the main server and if so redirects them. If the main server dies, the recovery server announces it to firewall so that it redirects the messages it receives to the recovery server.

Results

Everything up to the intermediate solution was implemented. Our application proposes safe communications as well as a safe storage and all this inside a basic graphic user interface. Regarding the advanced solution, we had to give up on the firewall. While it was mostly implemented, we found it complicated to integrate inside our solution. However, the recovery server is working, even though its functioning could be smarter. Communication between the two servers is not ciphered.

Evaluation

Our application is a bit lacking in user-friendliness in our regards. However, we quickly realized this should not be our main focus and therefore let it be that way. But, for example some weaknesses related to that would be: the possibility to have multiple people working on the same book at the same time without knowing it (meaning they might erase one another changes without noticing it) and the lack of basic features concerning basic text-writing on computer (typically the life-saving CTRL+Z). On the other side, we are proposing a solid graphic user interface which should allow any users with any level of “computer” knowledge to use our application efficiently and at small cost we allow authors to have the possibility to work on their books from any computers they want, wherever they want.

We reached our main goal which was to offer an application that proposes a place where authors can store their books without the fear of having them stolen. All our communications are ciphered and so is our storage. And even though users must have trust in us to really believe us as it is not possible for us to prove it, we have no way to actually access and read what they are sending to our server as it is encrypted in a way that do not let us any chance to decipher it. And finally, even if our server is taken over, attackers won’t be able to infer any type of information from the data stored. Passwords are stored encrypted in a secure fashion (with a cryptographic hash plus some salt), an attacker won’t therefore be able to retrieve them. Basically, any data stored don’t make sense without information stored in the client. And this is the core strength of our application.

The availability of our application is its main weakness. We are offering a cloud application that is very vulnerable to any type of denial of service attacks, as we did not implement sufficiently powerful tools to prevent them. We decided to put less emphasis on this aspect because, after discussion, we came to the conclusion this threat is very unlikely (why would anyone try to bring us down when we are still a nobody on our market) and it was therefore not a high risk to not try to prevent it better, whereas someone discrediting us by revealing

the data we store would have a huge negative business impact which makes it that much more of a risk.
We also assumed the main server and the recovery server are in a secure network, which allow us to have non-ciphered communication. We unfortunately had to make this assumption because of a lack of time. However, our main server can run fine without the recovery server, therefore if our assumption doesn't hold it would be better to run the main server without his recovery partner.

Conclusion

Overall, we are satisfied with our product. It works as expected, it is relatively stable, and contain some nice features. We are providing a cloud application with complex security mechanisms which is what we were asked to. However, we wish we would have had more time to test it more thoroughly.

References

The libraries we used to develop the project are the following:

- JavaFX for everything related to the user interface
- JDBC and mysql for everything related to the management of the database

Most of the readings we made to help us implement our application was on the lecture slides and on stackoverflow.com. One article that helped us set up the password storage can be found at the following address: https://nakedsecurity.sophos.com/2013/11/20/serious-security-how-to-store-your-users-passwords-safely/.