Security in Computing
Academic Year 2023-2024
Course Project: **Chat platform with end-to-end encryption**

Deadline 31 May 2024

Defence Same date as oral examination

Delivery Both the documentation and the code should be delivered via the project assignment form on Canvas in the form of a <name>-project.zip file on the day of the deadline **before 23:59**.

Material The project material consists of the assignment and a supporting Electron / NodeJS skeleton project. These are available on Canvas.

1 Assignment

The purpose of this project is to create a **secure** chat platform similar to Slack[1], which allows users to communicate messages in channels or as private messages. We provide a simple Slack clone that contains the main functionality from Slack, although without any regards to security, both at client side (Electron based) and server side (NodeJS based). The main requirement is to secure the implementation, limiting what data can be read by different parties (such as other users, or the server itself), and protecting it from common exploits. Exact details can be found below, along with the evaluation criteria. This project is open-ended and values original solutions and research.

2 Requirements

The server for the chat platform is written in NodeJS, and uses Socket.IO for communication. The front-end itself is a simple electron client utilizing HTML, jQuery, Bootstrap and Socket.IO. The goal of this project is to modify the provided skeleton to secure the platform.

There are some requirements w.r.t. security.

Login / Authentication. Right now users can authenticate as any user by just specifying a username. Extend the platform with proper logins and authentication.

Persistence. All messages are currently stored in-memory, in a JavaScript object. Modify the platform to persist these messages in some other way (to a file, Redis, MySQL, MongoDB,

[1] https://en.wikipedia.org/wiki/Slack_(software)

...). Whatever method you pick, you need to ensure that storage happens in a secure, non-exploitable manner. For example, if you would use an SQL server, you need to ensure that SQL injection is impossible.

End-to-end encryption. Implement some form of end-to-end encryption, ensuring that the server cannot read messages from users, that it is unaware of the message content. Furthermore, ensure that users can only decrypt messages that are directed for the user or channels the user is in. When a user is removed from a channel, ensure that the user can no longer read and decrypt new messages from the channel. Note: even if the removed user can sniff the network traffic containing messages to or from the channel, there should be no way for him to decode these messages.

Hint: This project heavily relies on the use of asymmetric encryption, used to exchange keys and data between the users. Do not forget that asymmetric encryption is very slow, so it should only be used to encrypt symmetric keys, that are then used to encrypt the actual messages or files. Think well about where those keys are stored, whether they are stored encrypted, and with which public keys.

Client-side security. The electron-frontend is currently not protected for attacks such as html injection. Ensure that proper filtering and escaping is applied to all input and/or output.

Dependency management. Remember that you are responsible for the security of the entire application, including any libraries it uses. Ensure that dependencies are managed and secured properly. For example, you may want to use integrity checks for client side libraries. At server-side you might not want to freeze NPM libraries to a certain version ...

Extra requirements (not obligatory, for bonus points):

- **Sandboxing.** Extend the platform to allow users to share 'dynamic' content with each other. This content should be encoded as small applications that can be shared, but where the access scope/capabilities for these applications are severely limited. For example, a user could share a small chess game that allows him/her to play chess with other users. The chess application itself should have no way to access information or modify behaviour of the chat platform, other than being able to communicate with the chess application running on some other users client. This could be added by means of sandboxing and/or capability frameworks [2].
- **External authentication.** Extend the platform to use an external authentication method (for example, OpenID[3]).
- **Perfect forward secrecy.** Add perfect forward secrecy to the platform. When a user's private key is leaked, it should not be possible to decode older conversations. Explain in your report specifically which extra security objectives these modifications achieve under which attacker model.
- **Toggle functionality for e2e encryption.** Implement the ability to disable end-to-end encryption on the platform. You can implement this as a configuration parameter on the server, disabling e2e encryption over the entire platform, or as an option when creating a new channel, disabling e2e encryption only for this channel.

Being creative and adding additional requirements of your own or any of the extra requirements to the project is appreciated and will be rewarded with bonus points. However, make sure that all of the basic requirements are implemented before starting on extra functionality.

[2] See resources.pdf for a list of options
[3] https://en.wikipedia.org/wiki/OpenID

2.1 Programming Platform

The project extensions are expected to be implemented in JavaScript but TypeScript and WebAssembly approaches will be accepted as well, if they are properly described and documented in the report.

2.2 Tips and hints

Basic encryption for non-e2e channels: the chat implementation provides three types of channels: public channels, private channels and private messages. As public channels are generally public, e2e encryption does not make sense there. You are not required to utilise e2e-encryption for public channels, but you need to at least ensure that all communication is encrypted with modern and secure encryption technologies.

Persistence of messages on clients: if you decide to persist messages on a client itself instead of on the server, ensure that these messages can be synchronised with other clients registered to the same user (i.e. if users login on multiple devices). Make sure that this happens in a secure fashion (e.g. the server cannot read the messages during synchronisation).

3 Testing and Report

You should evaluate your secured implementation, looking at interesting scenarios that show that your solutions are valid. These scenarios should show how your application behaves w.r.t. the different security aspects. Explain how your solution protects the platform against different attacks. For example, a scenario could be the ability to perform HTML-code injection in the chat, and you should describe how your implementation prevents this.

You will write a small report about the project. This report must be **no longer than 8 pages** (excluding figures) and serves as a guide for evaluating your project. The report should include:

1. Overview of your application and how the implementation fulfils the requirements.
2. Describe the security goals and attacker models for your application (remember: it is possible that different security goals apply under different attacker models). Think carefully about the attacker model and which components of your system are considered trusted.
3. Description of the important cases and design choices (w.r.t. security aspects) to consider for this project (e.g. how do you persist data, what methods do you employ for end-to-end encryption, etc).
4. Description of evaluation scenarios and behaviour of your application.
5. A small "manual" on how to run and test your application (e.g. if we need to install additionally dependencies or more. Ensure that the application can run out of your development environment). This manual should have been tested in a clean development environment on a computer other than your own to be sure it includes installation of all necessary dependencies etc.
6. Which nodejs libraries were added to the different parts of the application.

4 Evaluation

The project counts for 50% of the total course evaluation. Recall that you must hand in this project in order to pass for the course as a whole (as there is no partial evaluation).

What is **important**:

- Your evaluation (which you added to your report) should be focused on breaking the security of the application, and showing how your implementation resolves certain problems.
- In your evaluation, you should have carefully analysed security of your system in terms of an attacker model(s) and security goal(s).
- The project will be defended ***orally***, during this defence you will have to defend your design choices. Having a proper evaluation in your report will help you here. We might ask you to demo your application in some cases, for example if we failed to run it.
- Quality and structure of the code is **very** important.
- The project is to be made strictly individually and on your own. This means that you must create your project on an independent basis and you must be able to explain your work, reimplement it under supervision, and defend your solution. Copying work (code, text, etc.) from or sharing it with third parties (e.g., fellow students, websites, GitHub, etc.) is not allowed. Electronic tools will be used to compare all submissions with online resources and with each other, even across academic years.
- Any action by a student that deviates from the instructions given and does not comply with the examination regulations is considered an irregularity. Plagiarism is also an irregularity. Plagiarism means the use of other people's work, adapted or otherwise, without careful acknowledgement of sources. (cf. OER, Article 118$2). Plagiarism may relate to various forms of works including text, code, images, etc.
- Problems with the software tools and/or platform should be reported **no later than 8 May 2024**.

Please do not hesitate to contact me for further questions.
e-mail: jim.bauwens@vub.be, office: 10F716

Good luck!