CYBER SECURITY

SECURE MEDICAL SYSTEM

DUE TO 11/6

Alessandro Spinosi
Guillaume Kerckhofs
Samain Clément
Randi Dochot
Alix Declerck
Professor Romain Absil

Abstract. The goal of this project is to implement a secure client / server system handling patient's medical records, such as the ones available in hospitals.

We install un server using and existing tech combo (from Gerardo Fernandez) which is composed by docker to run the server, Nginx the server itself, Symfony 6.2 Boilerplate for the applicative framework and we change the database to mariadb

We create a client using npm run dev from the Node Package Manager

Contents

1. Project description

Installations procedures can be find here :
`https://github.com/Randi-Dcht/project_security_web`

1.1. Academdic hypothesis

In general the trust of a webbsite [1] is based on *domain* name. It's the information that will be asked by an authority like let's encrypt [2] and we don't have any domain name, this is an academic project. Therefore we will suppose that we grant to certification authority a domain name which is trusted.

A similar situation occur for the client which is iddentified by *email*, in some situation we want to be sure that the user is not an impostor (if Alice send a message to Bob we want to be sure it's not Oscar who might have slolen some information from Alice). We may consider that all parties have given one time the iddentity card and and confirmed their email addresss.

2. Check-list

2.1. Confidentiality

Non repudiation by certificate authority.

During creation, *patient* and *doctor* create their own public and private keys. The server activate a *certificate request* which is shaped in the Section 1.1 but we also use symfony [3] component that can text the user on mobile phone.

End-to-end encryption.

All data transmission are protected by end-to-end encryption. We use assymetric keys. When a *doctor* or a *patient* connect to the server we check by mail and phone if it's the right person. Then we send information for the meeting encrypted (using the public key). Then the *patient* and the *doctor* have to decrypt using they own private key which is then not stored on the server. Then *system administrator doesn't have any access to sensible data of some arbitrary user.*

2.2. Strong authentification scheme

Symfony [3] is implemented using *authentification* (it's the so called firewall system) and *authorisation* which involve access control to eveything.

- m.a.j todo

Do I properly ensure confidentiality?

- Are sensitive data transmitted and stored securely?
- Does a system administrator have access to the sensible data of some arbitrary user?

2.3. Integrity

- m.a.j todo

Do I properly ensure integrity of stored data?

- m.a.j todo

Do I use a proper and strong authentication scheme?

- Do I follow OWASP guidelines?
- Is my authentication broken (cf. OWASP 10) [4]

2.4. Relying secrecy

- m.a.j todo

Do my security features rely on secrecy, beyond credentials?

2.5. Injection vulnerability

Am I vulnerable to injection?

- URL, SQL, Javascript and dedicated parser injections

2.6. Remanence attack vulnerability

- m.a.j todo

Am I vulnerable to data remanence attacks?

2.7. Replay attack vulnerability

- m.a.j todo

Am I vulnerable to replay attacks?

2.8. Fraudulent request forgery vulnerability

- m.a.j todo

Am I vulnerable to fraudulent request forgery?

2.9. Monitoring

- m.a.j todo

Am I monitoring enough user activity so that I can immediately detect maliciousor analyse an attack a posteriori?

Do I simply reject invalid entries, or do I analyse them?.

Can logs be falsified?.

2.10. Component security knowledge

- m.a.j todo

Am I using components with know vulnerabilities?

2.11. System

- m.a.j todo

Is my system updated?

2.12. Access control

- m.a.j todo

Is my access control broken (cf. OWASP 10) [4]? Do I use indirect references to resource or functions?

2.13. General security features

- m.a.j todo

Are my general security features misconfigured (cf. OWASP 10) [4]? Also, note that you will unlinkely graduate should you fail to

use a (at least self-signed) certificate for your server,.

use a framework (at least for the server's side),.

achieve end-to-end encryption (if relevant)..

The end to end encryption is handled

References

[1] "Scamdoc." [Online]. Available: `https://fr.scamdoc.com/`

[2] "Let's encrypt." [Online]. Available: `https://letsencrypt.org/fr/`

[3] "Symfony." [Online]. Available: `https://symfony.com/`

[4] "Owasp guidelines." [Online]. Available: `https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/`

UMONS, Belgium
Email address: `alessandro.spinosi@student.umons.ac.be`

UMONS, Belgium
Email address: `guillaume.kerckhofs@student.umons.ac.be`

UMONS, Belgium
Email address: `samain.clement@student.umons.ac.be`

UMONS, Belgium
Email address: randi.dochot@student.umons.ac.be
URL: www.dochot.be

UMONS, Belgium
Email address: alix.Declerck@student.umons.ac.be

UMONS, Belgium
Email address: romain.absil@umons.ac.be