CYBER SECURITY

SECURE MEDICAL SYSTEM

DUE TO 11/6

Alessandro Spinosi
Guillaume Kerckhofs
Samain Clément
Randy Dauchot
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 improved a server and create a client to do a set of tasks in regard to the safeness principles like end-to-end encryption and certificates to avoid security issues that we have learned during the class of Professor Romain Absil at UMONS, Belgium

Contents

1. Introduction

Installations procedures can be find here :
`https://gitlab.com/Randi-Dcht/project_cybersecurity_lab3`

1.1. Framework versions

- The 'server' is developed using symfony (version 6.2), php (version 8.2) and maria DB (version 10.11.4) using an tech combo from Gerardo Fernandez [1]
- The 'client' is developed using node.js (version 20.2.0), npm (version 9.6.6), nvm (0.35.3)
- We chosen docker for practical reasons and we are aware that *podman* is not running in root which make this solution more safe. In practice all images used might being loaded in *podman*, we chosen docker because it's more practical to development phase.
- We have taken the last images to make the server's docker and the last version of node.js and supplementary modules for the client which is our approach to have an updated system.
- Important to notice that we carefully keep secrecy on softwares versions (php, ngnix, …) to hide the related weaknesses that are generally shown by versions.

Component security knowledge.

We checkt on the website as `https://www.acunetix.com`, `https://snyk.io/fr/`, `http://www.cvedetails.com` for node.js [2], symphony [3] and mariadb [4], and it appears that the named vulnerabilities were present until older version that the one we use regarding Section 1.1

1.2. Academdic hypothesis

In general, the trust of a website [5] is based on *domain* name. It's the information that will be asked by an authority like let's encrypt [6], 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, and we use a X.509 [7] self-signed certificate on our server.

A similar situation occur for the client which is identified 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 stolen some information from Alice). We may consider that all parties have given one time the identity card, and confirmed their email addresses.

2. Project description

As detailed in Section 1.1 handle a server (backend) and a client (frontend) architecture. Moreover, we manage 3 specific roles : *admin*, *patient* and *doctor*. We may refer as *patient* and *doctor* as a *user*.

2.1. Connection

We checked vulnerabilities of symfony [8] and not found our version into version that are affected by replay, timing, remanence attacks. Furthermore the attacker must have the right certificate and potentially access to the mailbox of the high jacked account.

Symfony [9] is implemented using *authentication* (firewall) and *authorization* (access control), and we think that is it indeed a strong authentication scheme.

Checking owasp [10] we also notice that most *authentication broken scenarios* don't concern us because we don't use any passwords to connect but the mail.

Inscription.

During inscription phases the server generate a *p12 file* which contains private key and certificate. The public key stays on the server. This certificate must be entered manually before login, otherwise the client can't access to the server.

All communications are done with end-to-end encryption.

The fact that *patient* and *doctor* own their own private keys and certificate which is shaped in the Section 1.2 is also a non repudiation warranty.

Medical reports access.

The *patient* can access their own files in database, all files are encrypted.

When the *doctor* have an appointment to a *patient*, the *doctor* can access the *patient* files. The *patient* send the *symmetric key encrypted* with the public key of the *doctor* which is consequently the only person able to decrypt this message with his own private key. It's only after this phase that the *doctor* can read the medical files of the *patient*.

When a *doctor* take action to update or delete on a specific file, the *doctor* send a message to the *patient* to do so. In practice, only the *patient* do destructive operations on proprietary files

To access it the *patient* and the *doctor* must have connected with a valid email and certificate which gives a nice secrecy beyond credentials and integrity.

2.2. Roles

A specific role mean some privileges, here we details

Admin role approve or revoke *user* and give *doctor* or *patient* role.

When the user is connecting to the server, his browser checks the certificate as explained in the Section 1.2 which grant the server authenticity.

- *Patient* and *doctors* have a private (own locally) and a public key (on the server).
- All files in database are encrypted with a symmetric key, as seen in Section 2.1.2. Consequently, the administrator role don't have access to medical files neither.

When the *patient* use a new device, the *patient* copy the *p12 file* given during the Section 2.1.1 on the new device, so both devices can be used in the same time.

Patient and *Doctor* can change credentials which mean change *p12 file* and change email which also generate new credentials because the certificate identification is based on the mail. We made that choose because the mail is something more reliable in terms of verifications than the name, for example.

The *doctor* can ask the *patient* a file's update or delete, and also stop a consultation relationship.

Patient can remove a *doctor* relationship, visualize, modify or delete his own files and grant access to the *doctor* to consult medical records.

For information selections, we use lists as follows : list of *doctors* for appointment, list of appointments with the *patient* for the *doctor*.

All the requests are indeed secured by the fact that we use certificate under the limitations we evoke in the Section 1.2.

2.3. Navigation

Forms.

We filter inputs using regexp to avoid parser injections using sql, web or javascript keywords.

Monitoring.

The server generate *logs* to let the administrator monitor all users activities and detect, analyze malicious events eventually. The logs stay on the server which make them more hard to falsifiable.

We take a model similar to a block chain. We construct a chain of logs using hashcode that make impossible to break the logs hour, number and comment to process a coherent hash for the next, etc.

2.4. Conclusion

This project was a wonderful team spirit web security challenge and we wanna thanks our teacher, Mr Absil [11] for this project and also the infinity wonderful course we receive at UMONS [12].

We learned to develop a website more securely and discover a secret world that appear to be more important than we can imagine before. Thanks.

Room for improvements.

When the *patient* or the *doctor* receive a consultation, we think that can be nice to send a message, with a tool like the symfony notifier [13] for example. Can be mails, sms, …

References

[1] "Boilerplate para symfony basado en docker, nginx y php8." [Online]. Available: https://youtu.be/A82-hry3Zvw

[2] "Sensio labs for node.js." [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-12113/Nodejs.html

[3] "Sensio labs for symfony." [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-11981/product_id-22402/Sensiolabs-Symfony.html

[4] "Sensio labs for maria db." [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-12010/Mariadb.html

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

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

[7] "X.509 certificate." [Online]. Available: https://www.openssl.org/docs/man3.0/man7/x509.html

[8] "Symfony vulnerabilities." [Online]. Available: https://symfony.com/blog/cve-2015-8125-potential-remote-timing-attack-vulnerability-in-security-remember-me-service

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

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

[11] "Phd romain absil." [Online]. Available: https://gitlab.com/rabsil

[12] "Umons." [Online]. Available: https://web.umons.ac.be/fr/

[13] "Symfony notifier." [Online]. Available: https://symfony.com/doc/current/notifier.html

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: randy.dauchot@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