**A Study of Security for Web Applications and APIs**

**A Degree Thesis**

**Submitted to the Faculty of the**

**Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona**

**Universitat Politècnica de Catalunya**

**by**

**Jaume Serrats Bayo**

**In partial fulfilment**

**of the requirements for the degree in Telecommunications Technologies and Services Engineering**

**Advisor: Jose Luis Muñoz Tapia**

**Barcelona, October 2018**

Abstract

The objective of this project is to create teaching material related to the different security areas applied to web development. The material consists of a theoretical explanation where the concepts are defined and some exercises where the concepts are applied to real-world examples.

The majority of proposed exercises are developed in NodeJS on Linux.

In this project we explore three different areas, which we considered were the most fundamental when learning the security fundamentals applied to websites

The first topic consists of an introduction / revision of basic cryptography. We do not deepen into the mathematical theory, but we focus on the practical applications of each concept.

The second topic is a review of the HTTP protocol and analysis all its vulnerabilities. Next we introduce the HTTPS and explain the several differences and improvements. We also explain in more detail on how public key cryptography is applied to HTTPS.

Finally, as the bulk of the work, we focus on the classic vulnerabilities we can encounter when developing a website, such as Cross-Site Request Forgery and Cross-Site Scripting. We have developed several examples and exercises in order to practice how to find these vulnerabilities in several simple websites and how to fix them.

Resum

L'objectiu d'aquest projecte és crear material didàctic relacionat amb els diferents àmbits de la seguretat aplicada al disseny web. El material està format per una explicació teòrica on es defineixen els conceptes, i uns exercicis on s'apliquen aquests conceptes a exemples reals.

La majoria d'exercicis proposats estan desenvolupats en NodeJS sobre linux.

En aquest treball s'aprofundeix en tres temes principals, que hem considerat que eren els més fonamentals a l'hora d'aprendre els fonaments de seguretat aplicada a webs

El primer tema consisteix en una introducció/repàs de criptografia bàsica. No aprofundim els conceptes matemàtics, sinó que ens centrem en les aplicacions pràctiques de cadascun.

En el segon es fa un repàs del protocol HTTP i s'analitzen totes les seves vulnerabilitats. A continuació introduíem el HTTPS i expliquem les diferències i millores. També expliquem en més detall com s'aplica la criptografia de clau publica al HTTPS.

Finalment i com a gruix del treball, ens centrem en les vulnerabilitats clàssiques que ens podem trobar a l'hora de desenvolupar una web, com són el *Cros-Site Request Forgery* i el *Cross-Site Scripting*. Hem desenvolupat diversos exemples i exercicis amb la finalitat de practicar com trobar aquestes vulnerabilitats en diverses webs senzilles i com arreglar-les.

Resumen

El objetivo de este proyecto es crear material didáctico relacionado con los diferentes ámbitos de la seguridad aplicada al diseño web. El material está formado por una explicación teórica donde se definen los conceptos, y unos ejercicios donde se aplican dichos conceptos a ejemplos reales.

La mayoría de ejercicios propuestos están desarrollados en NodeJS sobre linux.

En este trabajo se profundiza sobre tres temas principales, que hemos considerado que eran los mas fundamentales a la hora de aprender los fundamentos de seguridad aplicada a webs.

El primer tema consiste en una introducción / repaso de criptografía básica. No profundizamos los conceptos matemáticos, sino que nos centramos en las aplicaciones prácticas de cada uno.

En el segundo tema se hace un repaso del protocolo HTTP y analizan todas sus vulnerabilidades. A continuación introducimos el HTTPS y explicamos las diferencias y mejoras. También explicamos en más detalle cómo se aplica la criptografía de clave pública al HTTPS.

Finalmente y como grueso del trabajo, nos centramos en las vulnerabilidades clásicas que nos podemos encontrar a la hora de desarrollar una web, como son el *Cross-Site Request Forgery* y el *Cross-Site Scripting*. Hemos desarrollado diversos ejemplos y ejercicios con el fin de practicar cómo encontrar estas vulnerabilidades en varias webs sencillas y cómo arreglarlas.

To my parents, obviously, for everything

To the IT members of Telecogresca for introducing me to this world.

To Àurea, for all the support.

Finally, to Jose, for the advise and guidance.

Acknowledgements

Jose Luis Muñoz has contributed to the revision and improvements of the contents, and has also provided assistance with both NodeJS and LaTeX.

Revision history and approval record

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Purpose** |
| 0 | 20/09/2018 | Document creation |
| 1 | dd/mm/yyyy | Document revision |
|  |  |  |
|  |  |  |
|  |  |  |

DOCUMENT DISTRIBUTION LIST

|  |  |
| --- | --- |
| **Name** | **e-mail** |
| Jaume Serrats Bayo | Jaume.serrats9@gmail.com |
|  |  |
| José Luis Muñoz Tapia | jose.munoz@entel.upc.edu |
|  |  |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Written by: | | Reviewed and approved by: | |
| Date | 29/09/18 | Date | dd/mm/yyyy |
| Name | Jaume Serrats Bayo | Name | Muñoz Tapia, Jose Luis |
| Position | Project author | Position | Project Supervisor |

Table of contents

The table of contents must be detailed. Each chapter and main section in the thesis must be listed in the “Table of Contents” and each must be given a page number for the location of a particular text.

Abstract 1

Resum 2

Resumen 3

Acknowledgements 5

Revision history and approval record 6

Table of contents 7

List of Figures 9

List of Tables: 10

1. Introduction 11

1.1. Statement of purpose 11

1.2. Requirements and specifications. 11

1.3. Methods and procedures 11

1.3.1. Software 11

1.3.2. Documentation 11

1.3.3. Communication 11

1.4. Work Plan 11

1.4.1. Work Packages 12

1.4.2. Gantt diagram 12

1.5. Plan changes and incidences 13

2. State of the art of the technology used or applied in this thesis: 13

2.1. Topic 13

2.2. Topic 13

3. Methodology / project development: 14

4. Results 15

5. Budget 16

5.1. Equipment 16

5.2. Personal salaries 16

5.3. Total 16

6. Environment Impact (Optional) 17

7. Conclusions and future development: 18

Bibliography: 19

Appendices (optional): 20

Glossary 21

List of Figures

Fig 1 – Grantt diagram 12

List of Tables:

Table 1 Work Plan 12

# Introduction

## Statement of purpose

The objective of this project is to create useful resources in order to teach the fundamentals of cybersecurity applied to web design and development. It is aimed to students with a solid knowledge of the OSI model and some notions of web protocols and software development (specifically javascript).

## Requirements and specifications.

To complete the proposed examples and exercises it is needed:

* Computer with any Linux distribution (Ubuntu 16.04 and 18.04 were used)
* NodeJS v8.12 or higher
* Web browser (Firefox 62.0 and Chrome 68 were used)
* The following NodeJS libraries
  + Bcrypt
  + Express
  + Node-rsa

## Methods and procedures

Different procedures were followed depending of the aspect of the project

### Software

The software was developed entirely on NodeJS. This language was chosen because its raising popularity when developing a full web stack, allowing to program both back-end and front-end with only one language (javascript). Since this project was made with educational purposes, simplifying the software knowledge required to understand the content was an important objective.

### Documentation

The core of the project (found in the Annex) was written and formatted using LaTeX. LaTeX allows for easier formatting of a complex document. It was especially useful when applying syntax highlighting in the parts were code was inserted. It also allows better integration when used with Git, as explained next.

### Communication

Communication with the project tutor was done periodically and usually remotely. A VoIP software was used to communicate verbally, while a virtual machine with a VNC server was used to share a common screen. This setup allowed for more flexibility when setting up a meeting, as both our schedules were difficult to match.

## Work Plan

This project has 3 differentiate parts.

### Work Packages

|  |
| --- |
| Project: A Study of Security for Web Applications and APIs |
| Major constituent: Basic cryptography |
| Short description:  Understanding and developing examples that teach the basic fundamentals |
|

|  |
| --- |
| Project: A Study of Security for Web Applications and APIs |
| Major constituent: Web protocols |
| Short description: An introduction to HTTP, HTTPS, certificates and key management, HSTS and SNI. |

|  |
| --- |
| Project: A Study of Security for Web Applications and APIs |
| Major constituent: Web Security |
| Short description: Explanation of CORS, CSRF and XSS vulnerabilities in a website. Diverse exercises consisting of attacks on vulnerable examples. |

### Gantt diagram

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **WEEK** | | | | | | | | | | | | | | | | | |
| **March** | | | | **April** | | | | **May** | | | | **June** | | | | **July** | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| **TASK** | **Basic cryptography** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Web Protocols** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Web Security** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Figure

|  |  |
| --- | --- |
|  | |
| July | | | | August | | | | September | | | | October | | | |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
| TASK | Basic cryptography |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| HTTPS basics |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Web Security |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## Plan changes and incidences

Initially there was the intention to include a 4t block in the project about buffer overflows. This was dismissed during the critical review, since it was slightly out of scope, and also due to time constraints.

Another incidence that has affected the development of this project was the fact that I started working full time, greatly reducing the time available to develop the project. Because of this, an extension was requested in order to work the project over the summer.

# State of the art of the technology used or applied in this thesis:

Since this project is based on research, this will be a summary of the concepts learnt. **The full explanation can be found on the Annex**

## Basic Cryptography

### Symmetric cryptography

Symmetric key encryption use the same key for both encryption and decryption. In this project we’ve used AES, as it is the most standard algorithm used. We’ve also looked into block cipher modes and how they affect encryption of large blocks of data.

### Asymmetric cryptography

In asymmetric cryptography we have a pair of keys. A public key that is available to everyone and therefore is not secret, and a private key, that is only known to one user. The message encrypted with the public key only can be read with the private key, and with only the public key, it is impossible to know the private key. This technology is very useful in HTTPS as it has several interesting properties.

### Hash

A Hash function is used to map data of any size to a fixed size and makes very difficult to obtain the original o an equivalent input of the function knowing the hash result. This function is very useful when storing passwords, checking data integrity and digitally signing. In the Annex we also explain bcrypt, a hash function designed specifically to safely store passwords.

## Web protocols

### HTTP

HTTP is the base of the internet as people know. This protocol was not designed with security in mind and does not provide authenticity, confidentiality nor integrity. We have explained how a malicious agent could exploit the lack of these properties.

### HTTPS

HTTPS is a extension of HTTP that uses TLS encryption to add authenticity, confidentiality and integrity. To achieve this it uses a mix of symmetric cryptography (to provide confidentiality) and public key cryptography (provides authenticity). It is also required to implment a trust system called chain of trust to provide full authenticity. This is explained deeply in chapter 2.2 of the Annex

#### HSTS

Plain HTTPS is still vulnerable to downgrade attacks. The attacker performs a Man in the Middle and forces the connection to downgrade to plain HTTP. HSTS is a mechanism used to force the browsers to always connect via secure TLS.

## Web Security

Even if we use the latest protocols, languages and frameworks, there are several aspects that we have to be careful when developing the logic of a new website.

### Cookies

A HTTP cookie is a small piece of data that a server sends to the user's web browser. The browser may store it and send it back with the next request to the same server. Since HTTP is a state-less protocol, cookies are used to identify sessions and therefore are often the objective of web attacks. There are several parameters that a server can apply when sending cookies to the web browser.

### CSRF

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. First the attacker must design the URL or script, then trick the victim into activating it via social engineering or camouflage of the URL. These attacks are easily prevented using a CSRF token, as explained in the annex

### CORS

CORS is a mechanism used to allow browsers that applications in a domain access resources from a second domain. This action is blocked by default, and the second domain has to explicitly allow the browser to achieve this. Depending on

### XSS

The goal of a XSS attack is to inject malicious javascript code in a domain that the user trusts. Once an attacker achieves an injection, consciences can be severe, as the javascript can send information, modify the page, perform actions in the site, etc.

XSS attacks can be classified between front-end and backend, and reflected and stored.

If the XSS occurs when the server sends HTML with malicious code injected in it by an attacker, we'll classify it as Server XSS or Back-end XSS, when the injection occurs on the DOM (Document Object Model, the interface presented by the browser to interact with the HTML in the page) we'll classify the vulnerability as a Client XSS or Front-end XSS.

Persistent XSS occurs when the attacker inserts the payload in a persistent database in the application (usually in the back-end, but can also be in the front-end in a HTML5 local storage.). This type of attack affects all the users that request the affected content and is considered the most dangerous. Reflected XSS happens when the user input is returned immediately (search query, form, etc) and the input is not protected. This usually requires some sort of social engineering attack to be effective (tricking the victim to click a URL with malicious GET parameters, suggesting the victim to introduce a payload in a form ...)

There are two main protections agains XSS, validation and encoding. Validation consists in analyzing user inputs and actively removing the malicious code. Encoding escapes the inputs so they can be safely sent to the browser. Those protections are not mutually exclusive and they both should be applied correctly for maximum security. A deeper explanation can be found on chapter **3.4.4 - XSS Prevention** on the annex.

# Methodology / project development:

## Software development

All software used for the examples and scenarios has been developed on NodeJS 8 using the WebStorm 2018 IDE. This IDE is meant to be used to develop websites front-end, but since NodeJS is javascript-based, it can also be used with this language.

We’ll now explain how the software for each chapter was developed.

## Basic Cryptography

The main challenge for this chapter was learning to work in the asynchronous nature of NodeJS. Having plenty of experience with object oriented and functional programming, adjusting the mindset to asynchronous programming was quite difficult.

We did extensively use the crypto library of the NodeJS standard library. The examples found in the documentation were good, but only the simplest methods were documented. It was difficult to find information about the deep functioning of the library.

We also used the bcrypt library in the hash section. This library implements the brcypt hashing algorithm for passwords in NodeJS. In this case the documentation was very clear and presented no problem.

## Web protocols

This chapter was mostly theoric. That is why the only scenario developed on this chapter consists on two HTTP servers to test CORS. The first website is very simple and only consists on a express page that returns a static HTML file. Express is a popular NodeJS framework for programming websites back-end easyly. It is very easy to deploy a simple site.

The other site consists on another express web server, but has cors protection enabled. We enabled cors on the server with the library *cors*. It is very simple to enable the default protection, with just one line of code it’s set.

## Web Security

In this chapter there are several exercises and examples developed both to find the vulnerability and to fix it.

All the exercises consist on a express NodeJS web server as a back-end that serves one or more HTML pages.

The exercises that form the “XSS game” were adapted from an open-source course from Google. The original exercises were developed on Python, and migrated all the back-end code to NodeJS to fit the rest of the project.

# Results

At the end of the project we have obtained several scenarios that will hopefully be valuable at understanding the fundamental principles that have to apply in web design in order to have a secure site. The full documentation created, the scenarios and examples can be found on the Annex

# Budget

Since this project is based on software development, no hardware or prototype was required. All software used is open source and free to use.

Because of this, this budget will only contain the costs of the computers on which the project was developed and the time spent.

## Equipment

This project was developed in its entirety on:

|  |  |
| --- | --- |
| Item | Price |
| Laptop Dell Latitude 3340 | 430€ |

## Personal salaries

Assuming we rate the cost of a junior software developer at 10€/h and this project is valued at 18 ECTS credits (28h per credit)

* 10€/h \* 18 ECTS \* 28h/ECTS \* 1,14 (approximate tax) = 5745.6€

## Total

|  |  |
| --- | --- |
| Concept | Cost |
| Equipment | 430€ |
| Pesonal | 5745.6€ |
| TOTAL | **6175.6€** |

# Conclusions and future development:

As internet gets involved in more aspects of our daily life, the need for safe security practices arises. Websites and applications store all kinds of information that if fallen in the wrong hands could be very harmful.

After doing the research needed for writing this project, it’s become clear that even though modern protocols are much safer than before, it is important to always keep in mind safety when designing any software that is connected to the internet.

That is why is very important to educate properly on security matters to every engineer and technician involved in the design and deployment of any system that deals with sensible information.

Bibliography:

1. *Transport Layer Security (TLS) Extensions*. IETF RFC 3546, June 2003.
2. *The Transport Layer Security (TLS) Protocol*. Version 1.2. IETF RFC 5246, August 2008.
3. "Net | Node.js v10.11.0 Documentation". [Online] Available: https://nodejs.org/api/net.html [Accessed: 20 September 2018].
4. "Crypto | Node.js v10.11.0 Documentation". [Online] Available: https://nodejs.org/api/crypto.html [Accessed: 21 September 2018].
5. "bcrypt – npm | Usage". *Tetcos* [Online] Available: https://www.npmjs.com/package/bcrypt#usage [Accessed: 10 June 2018].
6. "Introduction to the DOM”. [Online] Available: https://developer.mozilla.org/en-US/docs/Web/API/Document\_Object\_Model/Introduction [Accessed: 13 July 2018].
7. "Express routing”. [Online] Available: http://expressjs.com/en/guide/routing.html [Accessed: 13 July 2018].

Glossary

* RSA - Rivest–Shamir–Adleman
* AES - Advanced Encryption Standard
* SHA - Secure Hash Algorithm
* HTTP - HyperText Transfer Protocol
* HTTPS - HyperText Transfer Protocol Secure
* CORS - Cross-Origin Resource Sharing
* CSRF - Cross-Site Request Forgery
* XSS - Cross-Site Scripting
* JS - Javascript