**SECURITY AUDIT**

**Name: Sarkis Chichkoyan**

**Student ID: 9827770**

[Introduction 2](#_Toc89445747)

[Audit method 2](#_Toc89445748)

[Audit results 2](#_Toc89445749)

[Conclusions 7](#_Toc89445750)

[References 8](#_Toc89445751)

|  |
| --- |
| **Report** |

# Introduction

The goal of this report is to show the vulnerabilities of a website. We were given a source code to analyse. We must make an audit and point out what are the issues inside the system using different methods. First, we will look at the different method used to find the security issues and why we chose them, then we will talk about all the results that we got and then focus on one of them and explain how we can prevent it from happening.

# Audit method

First the logical thing is to go through the code manually and try to figure out where the issues can be inside the source code.

Then I used bandit method is a static code analysis, that means it goes through the code and displays a list of the different issues found, if there is one it shoes the chunk of the code that is the problem and then gives us the importance of the risk (low, medium, high or undefined).

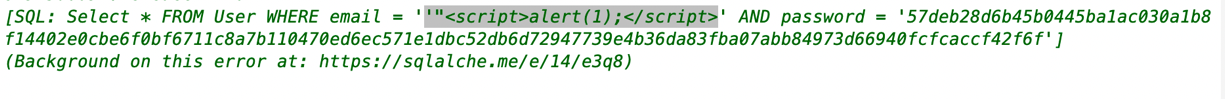
Another method that I chose is the OWASP ZAP software which will scan every page of the website, that means we must run the website first. It is a more precise way of searching the errors because it goes through every method used by the pages and displays all the alerts in a window. When we click on an alert we can see where it occurs and how many times it appears. Same as bandit it will tell us the importance of the risk, but also point out the attack and the evidence with a description of what the problem can be and a proposition of solutions that could avoid the issue.

I chose those two methods because one of them is analyzing the code when it is not running and the other one is more elaborate and analyzes deeply into all the links of the website when it is running.

# Audit results

**Cross side scripting:**

First, cross site scripting, in this screenshot we can see the attack underlined, the attacker is able to execute the user’s code. It will allow the hacker to access sensitive data by the browser. The main problem with XSS is that the user will not want to put his information on the website. To be sure that the attack will not be persistent we must change quickly the error inside the code. The risks are that if the attacker has access to the user’s data, then it will be easy for example to send emails with malicious links that will possibly extract more data for them. This issue is a high risk one, so we must be very careful to not let it happen.

****

**SQL injection:**

Une image contenant texte

Description générée automatiquementUne image contenant texte

Description générée automatiquementThe SQL injection issue is found by both bandit and OWASP ZAP. We can see that the password is protected because it is being hashed but the email part seems to be the error. The “(” is not normal to be there and may allow SQL injection to be made. To be able to avoid this problem we must allow only certain characters creating an “allow list” because all the data entered must be checked.

**Buffer overflow:**

Une image contenant texte

Description générée automatiquementBuffer overflow occurs also in this site, it overwrites memory space of the background process, and it can lead to multiple problems. It can be attacked by hackers by sending data to a program then the information is overwritten. The data sets the value of the return pointer so that when the function returns, it transfers control to malicious code contained in the hacker’s data. It can produce also crashes or even put the site on an infinite loop so that the site is no longer available. One of the main solutions to avoid this is to use a length checking function so that we are sure we are never going to get overwriting.

**X-Frame-Options Header Not Set:**

Also found an X-frame-options header not set that can lead to clickjacking attacks. Basically, clickjacking attacks consist of making the user click on an element or a link that hides something like a malware or malicious websites or even take sensitive information from the user. This can be done using an invisible page or element that would be implemented on the website and easily hack a user. To avoid this type of attacks the first thing to do is to use X-frame-options and be sure that it is set on all the pages.

**Insecure usage of temp:**

Bandit found us another error which has a medium risk, and it is insecure usage of temporarily files inside the meta.py file. This issue is an easy access for an attacker because it can allow the creation of a file with a predicable name before the site does and then other files could be accessed and lead to deletion or modifying them. And here the problem is that if they access the logging.log file they could have information on the users’ emails and passwords.

Une image contenant texte

Description générée automatiquement

**Absence of Anti-CSRF tokens:**

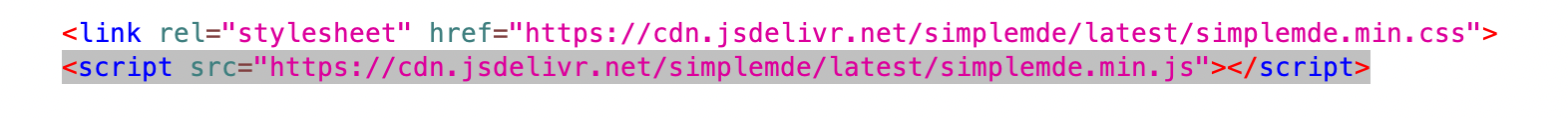
The absence of an anti-CSRF token is a problem that we can find in our website. It can lead to Cross-site request forgery attack which will give the hacker an access to execute an action as another user, and that can be changing a username, an email and a password to take control of the account. The attacker can also maybe create another admin user and then access all the private data of the website.

Une image contenant texte

Description générée automatiquement

**Cross-Domain JavaScript Source file inclusion :**

We always must be aware that third-party domain can be dangerous for our website because we can’t manage it. And the most recurrent problem is that if this cross-domain file contains malicious content it can be used against the website. To prevent these types of errors, the easiest would be to not use them but if we use them get well informed on what we import into our system.



**Security vulnerabilities discovered:**

|  |  |
| --- | --- |
| **Issues** | **Risk** |
| Cross site scripting | High |
| SQL injection | High-Medium |
| Buffer Overflow | Medium |
| X-Frame-Options Header Not Set | Medium |
| Insecure usage of temp | Medium |
| Absence of Anti-CSRF Tokens | Low |
| Cross-Domain JavaScript | Low |

Discussion of chosen vulnerability

Like we saw in the previous section there are many vulnerabilities on this website. I chose to talk in depth into one of them which is cross-site scripting. On this case we have reflected cross-site scripting which is not supposed to be a major risk for the website. But it can become one if we don’t deal with it. A reflected XSS occurs when the website receives a HTTP request, at that moment the attacker can change the request and the user will be redirected to that link. And that link could be a malicious link and lead to problems for the user.

In fact, the impact can be very dangerous, because if the hacker can execute what a user can then he could compromise the user. He could perform all the action that the victim can access. That means take sensitive data from them for example.

To avoid this problem to occur on our website we can use encoding so that the data is not readable for the attacker. Also, if the user receives emails from unknown people that could be because of reflected XSS so do not click on these. Users must ignore messages from unknown people for example on a social media.

Another solution could be to import web application firewalls like “impreva” for example. The OWASP ZAP software says that it is a high-risk issue and I think it is totally defendable because we saw how dangerous reflected XSS can be for the victim user and even the admin user. XSS is top 7 of OWASP from 2017 but previously it was top 2 and top 3 (2010-2012).

I searched for a real-world example to make it clearer. And in December 2016, a user found that Uber had a reflected XSS problem. And only two years later a response came in from Uber service which was:

“Due to a lack of input validation from the search field on lert.uber.com, it was possible to obtain a Reflected XSS from the URL path, e.g., https://lert.uber.com/s/search/All/Home">PAYLOAD.

Thanks, @hussain\_0x3c!”

This proves that every website and system can be touched by those attacks.

# Conclusions

To finish, we saw how to make an audit of a website by using different methods to be able to identify some of the vulnerabilities. Of course, to be more precise, we maybe must use more than two methods depending on the time that you must make the audit. Another conclusion that we can have been that analysis without the site running are not as precise as when the site is on. Making this audit also proved that we must be careful when we deal with security programming a website, because it is not only us that we put on danger but also all the users that potentially will us it.

|  |
| --- |
| References |

<https://cheatsheetseries.owasp.org/index.html>

<https://portswigger.net/web-security>

<https://github.coventry.ac.uk/pages/aa9863/6005-CEM/2021-22/>

<https://beaglesecurity.com>

<https://owasp.org/www-community/attacks/>

<https://hackerone.com/reports/191810>

<https://medium.com/@cxosmo/owasp-top-10-real-world-examples-part-2-3cdb3bebc976>

OWASP ZAP: version 2.11.0 - 2020-2021

Visual Studio Code: version 1.62.3

Python: version 3.8