6005CEM Security

Security Audit Coursework 1

Student ID: 9564473

Table of Contents

Introduction

This is a web application security audit report about the vulnerabilities found when using various auditing methods through crystal box testing and the implications that the vulnerabilities have on the website including the legal factors in may face.

Part 1 : Audit Methods

Audit methods are ways we can find vulnerabilities through procedural steps; it's done to ensure that a system put in place is working as expected and in the guidelines of the ethical standards they may follow, like for UK/ European companies that must follow GDRP guidelines. The focus of the GDPR audit is to determine whether the organisation has implemented adequate policies and procedures to regulate the processing of personal data. (eu gdpr institute, 2023). Crystal box test is when the person auditing has access to the source code of the system. On onsec.io, it is described as 'Crystal box testing is a type of white box testing in which the tester is provided with access to the full source code and documentation, as well as being given in-depth knowledge of the application, servers, and infrastructure' (ONSEC.io Research Team, 2023). There are three main ways to conduct the security audit when you have full access to the system: Manual code review, Automated source code Scanning and Automated application scanning (Malik, 2023). In terms of this audit, all three methods were used.

Manual Code Review

Manual code review is an important part of completing a security audit when you have full access to a system because it is mainly used when automated tools like source code scanning miss out on issues an experienced developer or peer might catch. The purpose of it in terms of security is to improve code quality and security when compliers and testing fail to find these issues.

The main strength of manual code review is that it can find vulnerabilities no tool or method can find. It allows the auditor to be able to analyse the logic behind the code where they can get a good understanding of the potential security issues. It is also a great way for people to learn and understand code.

The problem with doing a manual code review is that it is time-consuming, and even experienced developers can take long amounts of time to fix issues. It's also unlikely you'll find the same issue more than once.

Overall, manual code review expresses the importance of knowledge when conducting them as it can be very time-consuming; the person conducting it needs to understand the logic behind the code.

Automated code review

Automated code review is the process where an automated tool goes through the code to find common code errors or security vulnerabilities. Its process follows set guidelines and is usually integrated into the development process. The main purpose of it is to use static analysis to look for these potential issues.

The main strength of an automated code review is that it's very efficient and saves time when finding common code issues, and, in some cases, there are also tools that can do continuous inspections of the code. Another benefit would be that automated tools integrate very easily into the development process and are effortless when scaling up.

Automated application scanning

Automated application scanning is when the auditor uses an outside tool to find security vulnerabilities like cross-site scripting, injections, or insecure server configurations. It's the process of testing the application in the operating state. Tools like Burp Suite and OWASP Zap are referred to as dynamic application security tools (OWASP, 2023).

The advantage of automated application scanning is its ability to produce fast results from complex vulnerabilities; the other benefit would be that there is constant monitoring where it scans the application at different stages to give complete and thorough testing.

The disadvantages of automation scanning would be the false positives, and not all vulnerabilities will be found. The secure layer website discusses this scanner using a threat database, which can only detect vulnerabilities that have already been identified and documented in the database. It does not cover those yet to be discovered and thus cannot protect you from new threats' (securelayer7, 2023). We could also say these tools need to stay up to date to make sure they know about any new security issues.

Part 2: Audit Results

This section of the report is where I discuss my findings; it explains the process taken to find these security issues along with the method I used to find them. This section also discusses the implications of security vulnerabilities and the legal factors that can come from them.

Manual Code review Issues

Unauthorised Review Update

The first issue I found through manual code review was that in the source code access the review update section without being logged in. I could go into the URL and type, as seen in Figure 1; this allowed me to leave a review by checking who was accessing this page. It didn't check for any cookie session to authenticate that I'm the user leaving this review. In Figures 1 and 2, you'll set the page I was able to access review page as user 1 in the database. In figure 3, I was able to leave a comment as that user.

```
127.0.0.1:5000/review/1/1
@app.route("/review/<userId>/<itemId>")
```

Figure 1 REVIEW URL

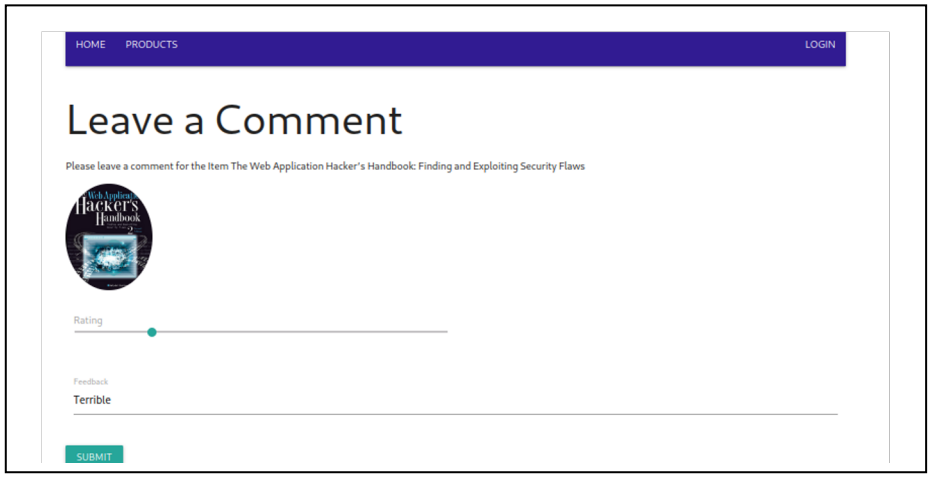

Figure 2 Review page

Figure 3 Review

This is a serious security issue because it gives unauthorised users access to a section of the website that should only be accessed by authenticated users. This is high risk because it allows modification of the review section without the actual user's knowledge, which means the integrity of the site is compromised.

The implications of this in the long term are that user trust can be lost, and the attacker can exploit this in promoting their product which will also harm the competition.

To prevent this, I would have the route check for an active session that would be set when the user logins before allowing accessing to a apart of the site.

Password Update without Authentication

The next Issue, I found through manual code review, was that I was able to update a user password when logging in, same as issue one, it doesn't check for an active session before letting me in through URL. In Figure 4, you see the URL I used to access the user setting where I was able to update the password.

The implication is severe as attackers can obtain the account of a user and get all their personal information without authentication, like the email in Figure 5.

127.0.0.1:5000/user/1/settings

Figure 4 Settings URL

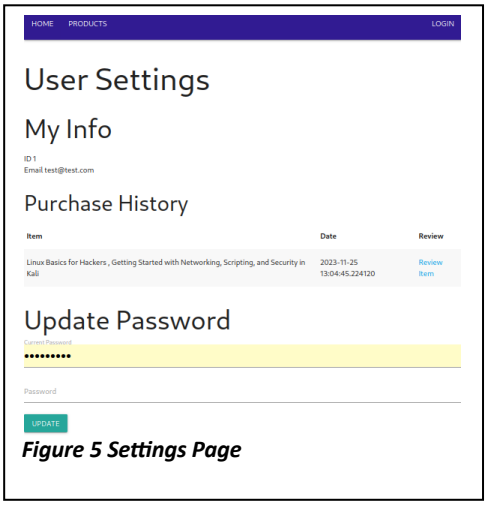

Figure 5 Settings Page

To address this issue, I would have the session management included in this section as well as have 2FA or MFA implemented in this area to prevent attackers from changing information or even accessing any user account.

Cross-site Scripting

The next issue, I found back in the review input field is that you can cross-site script within it. In Figure 6, you'll see the script I used to get the alert when the user opens that product. The implication of cross-site scripting is that session hijacking and data theft can be done to steal user information like cookies and login credentials.

By implementing input validation and sanitation, you can prevent these sorts of threats in the input fields. It could also have security headers like x-content-type-options.

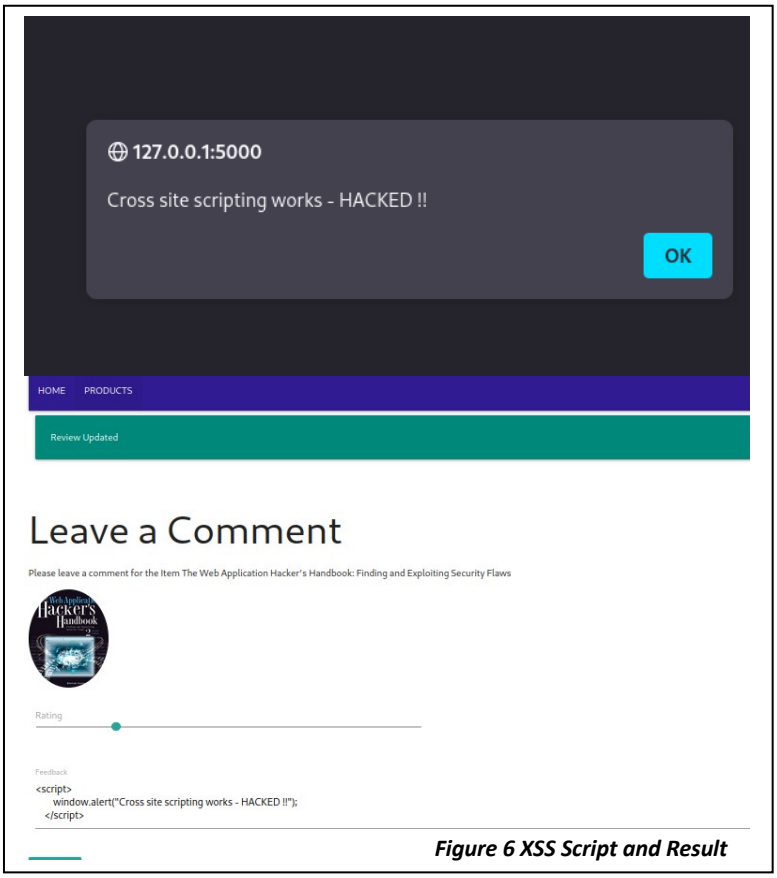

Figure 6 XSS Script and Result

Unhashed Passwords

The next issue I found through manual code review was in the SQL database and the views.py file in Figure 7; none of the passwords are hashed and salted, and the input that the user gives is directly stored in the database. The implications of this would be severe; if there was a data breach, it would comprise sensitive user data. I would implement a Bcrypt hash and combine that with salt, then store that hashed input in the database.

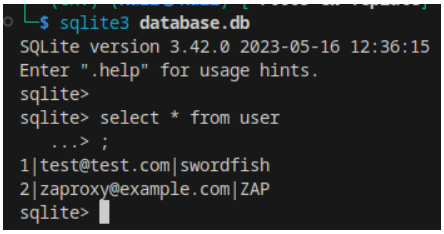

Figure 7 Database Passwords

Missing Terms of Service File

The last issue I found from manually looking through the code was from the create an account section in Figure 8. There was no file for the terms, but there was a link the user could try to access it. The Implications of not having terms and conditions would be severe from a legal standpoint as it could lead to dispute, loss of trust and legal issues. In terms of GDRP users will need to know they consent to the personal information they share on the website; they will need to know it won't be shared. I would include a term of service to ensure users of transparency and legal compliance.

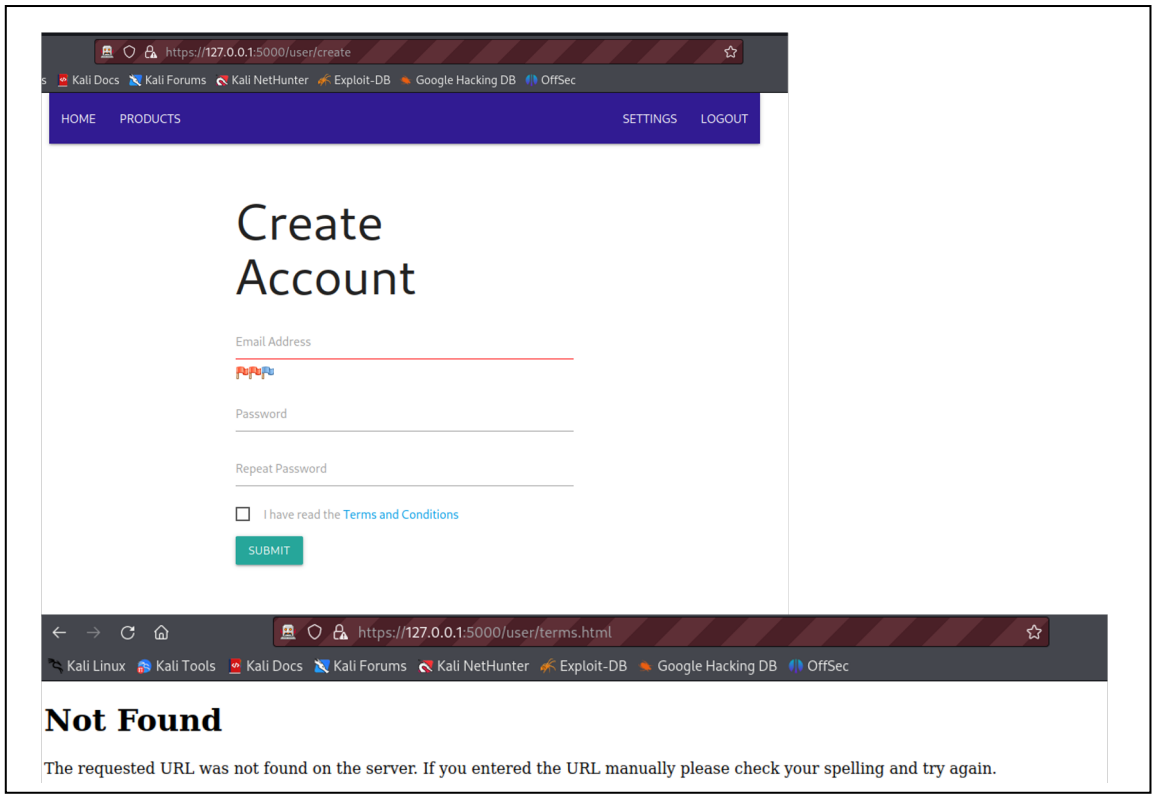

Figure 8 Create an account page and Terms page

Automated Source Code Scanning Issues

Hardcoded Password in Source Code

The next vulnerability I found was within the meta.py file, which you can see in Figure 9, where a hardcoded password was found. Bandit automated scanner detected sensitive passwords directly in the source code. The implications of this are that anyone who gets access to the source code will have access to any sensitive information. To fix this, I would utilise secret management to store and take sensitive information.

```
app.config.update(
   (SECRET_KEY = "secr3t!"),
   (SESSION_COOKIE_SAMESITE = "Strict"),
   (UPLOAD_FOLDER = UPLOAD_FOLDER)
```

Figure 9 Secret key

Automated Application scanning Issues

The next tool I decided to use was the application scanning tool because it would evaluate the web application while it was running to find some of the more complex issues.

SQL Injection vulnerability

The next issue I found was that SQL map was able to find a vulnerability within the login field where it could do an SQL injection and then get the contents of the User table within the database. In Figures 10 and 11, you'll see commands and results used to dump all the data found in the SQL database. Like earlier in the document, I would implement input validation and try avoiding dynamic SQL queries.

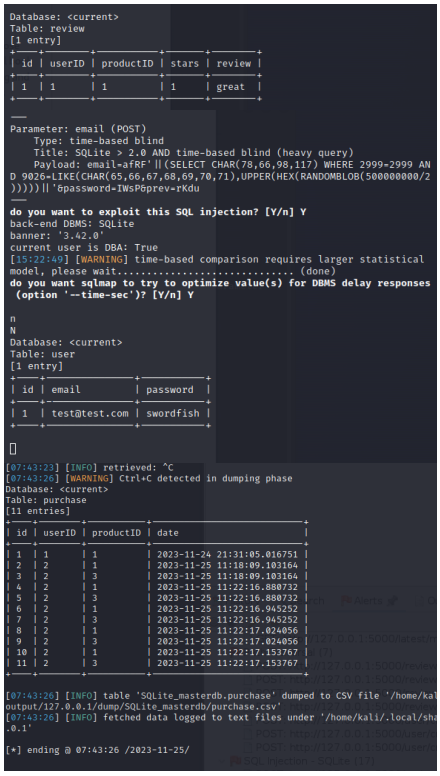

Figure 11 SQL --DUMP

sqlmap -u "http://127.0.0.1:5000/user/login" --data="email=test@test.com&password=swordfish" --dbms=sqlite --dump --threads=8

Figure 10 SQL MAP COMMAND

Brute Force Vulnerability

The next issue I found was from using Burp Suite to brute force into the login using a sniper attack; from when I found the update where the user email was test@test.com, I was about to do a brute force word list attack to find the password. The implications of this are that it could allow unauthorised access to an attacker. To tackle this issue, I would implement a policy which would have a few attempts and have 2FA (2 factor authentication) just in case an attacker was able to get the password.

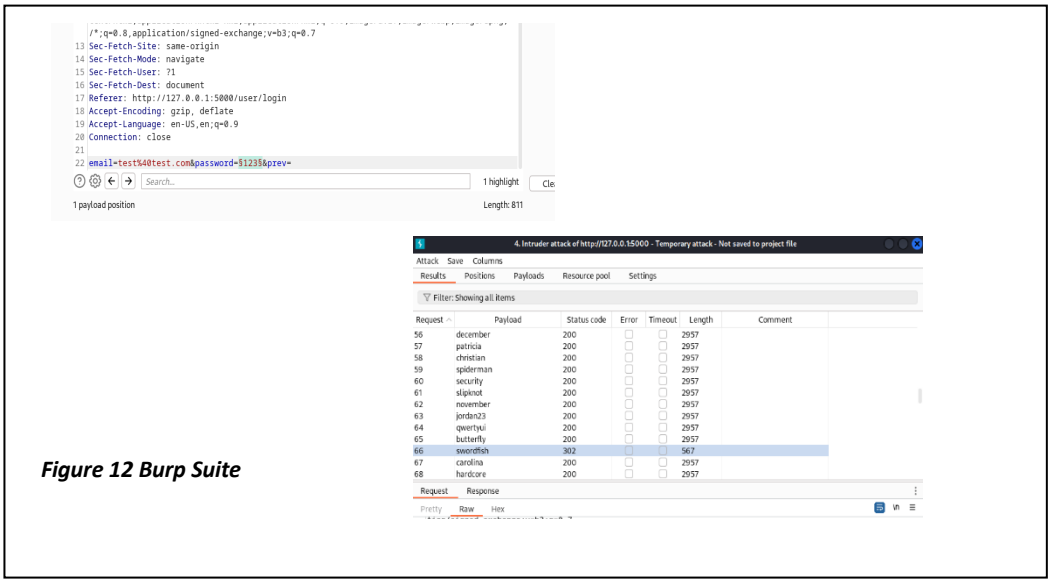

Figure 12 Burp Suite

Conclusion

In conclusion, this security audit outlines the poor security practices from unauthorised access, poor session management, vulnerabilities that allow cross-site scripting, lack of password management and the lack of compliance with GDRP. The Audit Provide the best security suggestion to improve each one of these issues that also comply with GDRP laws and the OWASP practices. It's urgent that all the implementation practices are put in place to improve the web application's defence against any attack.

Bibliography

eu gdpr institute. (2023, 11 25). *GDPR Audit Approach*. Retrieved from eu gdpr institute: https://www.eugdpr.institute/gdpr-audit-approach/#:~:text=Detect%20data%20breaches%20or%20potential,policies%2C%20procedures%20and%20IT%20platforms.

Malik, K. (2023, 11 25). *Code Review: Manual VS Automated*. Retrieved from codiga: https://www.codiga.io/blog/code-review-manual-vs-automated/

ONSEC.io Research Team. (2023, Jan 12). *White, black, gray, or crystal?* Retrieved 11 25, 2023, from onsec: https://blog.onsec.io/white-black-grey-or-crystal-all-kinds-of-penetration-tests-explained/

OWASP. (2023, 11 25). *Vulnerability Scanning Tools*. Retrieved from OWASP: https://owasp.org/www-community/Vulnerability_Scanning_Tools

securelayer7. (2023, 1 18). *The Pros And Cons Of Vulnerability Scanning Tools*. Retrieved 11 25, 2023, from securelayer7: https://blog.securelayer7.net/vulnerability-scanning-tools-pros-and-cons/