

1910716 Database and application development - final

by Ollie Bird

FILE	18246_OLLIE_BIRD_1910716_DATABASE_AND_APPLICATION_DEVELOPMENT_-_FINAL_317166_1891290829.DOCX (27.54K)		
TIME SUBMITTED	10-MAY-2021 12:51PM (UTC+0100)	WORD COUNT	1640
SUBMISSION ID	152484002	CHARACTER COUNT	8802

Databases and Application Development – Application

ACCA5026

1910716 – Oliver Bird

Apprentice Software Developer – BT Digital, Data, Data Services

1910716@student.uwtsd.ac.uk

Note – To access the admin page the file is cardiffCheeseCoAdmin.php. Required user role within the database is 2.

ABSTRACT

This report analyses the application layer security of the Cardiff Cheese Co's (CCC) website. The report contains a comprehensive, Whitebox analysis of the site following modern standards. The report contains a number of current and largely futureproofed solutions to maximise the security of both the host website, and its users.

Table of contents:

- 1. Security Outline**
 - a. OWASP**
 - i. Application Security Risks
- 2. Current Protections**
- 3. Breaches / found exploits**
- 4. Pre – deployment Fixes**
 - a. Static Application Security Testing**
 - b. SSL Certification**
 - c. Web Application Firewall**
 - d. Limit user inputs**
 - e. Database management**
- 5. Active Protection**
 - a. Dynamic Application Security Testing**
 - i. Zap**
 - b. Active Penetration tests**
- 6. Appendix**
- 7. References**

- 1. Security Outline**
 - a. OWASP**

OWASP (Open Web Application Security Project) is a large open-source community project working to improve web security globally. A number of tools and guidelines will be suggested below.

i. Application Security risks

OWASP maintain a list of what is believed to be the top 10 most critical lapses in security of the modern web application [1]. Each will be analysed with regard to the CCC site where appropriate.

- Injection

The case in which a user may manipulate the input given to the backend [A] interpreter in the form of additional querying. This can cause the interpreter to execute further commands than intended when built and reveal protected data inside of the database.

- Dysfunctional Authentication

Authentication methods can potentially be bypassed by an attacker if pages are left unprotected or with minimal authentication. The authentication methods used must be robust enough to accurately affirm a user's identity both before and during browsing.

- XXE – XML External Entities

This weakness is not prevalent within the CCC site as all files are contained locally.

- Broken Access Control

User access must be limited and enforced correctly on all pages of the site. Functionality and access should follow the principle of least privilege [B] in all cases.

- Security Misconfiguration

All aspects of the website must individually be made secure. This includes external libraries, error messages, HTML etc. Due to the CCC site being coded from scratch, using no external libraries or content, this type of weakness has little prevalence.

- Cross Site Scripting – XSS

The inclusion of user given data within a webpage. It manifests when a user supplies a script to the page which creates another, using the supplied malicious code.

- Insecure deserialization

This type of vulnerability does not affect this site as no user data is deserialized at any point. No data is reconstructed or reconstructed anywhere on the site.

- Using Components with known vulnerabilities

No components are used within the CCC site, therefore this component-based issue is not present. Any code written into the sites front and backend were hand built from scratch, using no imports or external libraries.

- Insufficient logging and monitoring.

The most prevalent issue affecting the CCC site. No logging has been implemented for both login and user creation, more so no limitations have been created for these mediums. The current infrastructure of the site has no active monitoring tools enabled.

2. Current Protections

The pre-deploy version of the site has a number of basic security features:

- Prepared statements

All user inputs are sanitised before being sent to the backend via prepared statements. This protects largely against SQL injection attacks by creating the query in two parts, without directly allowing user input to be injected into the interpreter.

- User authentication

All administrative functions are hidden behind an admin portal in which can only be accessed via a role assigned in the database. User role is not affected or assigned via session variable, meaning only a database query can determine

the role of a user. This prevents malicious acts in which an attacker attempts to spoof the session details of another user, as when accessing any restricted page, the role is checked.

- **Input sanitisation**

Several user inputs are limited in the html as well as within the database. This limits the lengths of user inputs further to limit SQL injection.

- **Honeypot**

User inputs have been created with a hidden field. The php controlling these inputs has been designed to redirect bots away from these pages without passing the information to the database.

3. Breaches / found exploits

Without the use of advanced attack tools, a determinate list of current vulnerabilities is difficult to complete.

The site has two large weakness points:

- Login
- Create User

The login currently has no protection against a brute force attack [C] against the 'userdetails' table. This means a simple Username / password cracking tool could break all users on the site. No monitoring is currently implemented to combat this type of attack, and ultimately would root [D] the site.

The create user form also has no form of limitation on creation of users. This can cause mass addition of users into the database and potentially crash the backend taking the site offline.

Additionally to these two vulnerabilities, there is no authentication or monitoring of user actions within the site. The use of SSL or a traffic manager [E] have not been implemented to protect and monitor connections to the site.

4. Pre – deployment fixes

Prior to the deployment of the site onto the internet, further protections must be added to ensure the protection of the site and its users.

- a. Static Application Security Testing (SAST)**

SAST is a pre-deployment security test of the base source code of the site.

The code is parsed to identify known and potential vulnerabilities via an automated tool(s) [2].

b. SSL Certification

Upon the site going live, HTTPS connections must be enabled via the addition of an SSL certificate. The certificate should come from a known vendor (not self-signed) to ensure confidence in the site. This will allow all information passed to the site to be encrypted, meaning information cannot be sniffed when being passed over the internet. This will also help to reduce the chance of a 'man in the middle' style attack, as the root certificate is created via an approved vendor. HTTP should be disabled, and all users should be redirected via error 301 to the HTTPS equivalent.

c. Web Application Firewall (WAF)

A WAF should be implemented to protect and monitor incoming traffic to the site. This firewall can redirect known botnet traffic and track user actions within the site. The WAF will keep record of potential malicious users and restrict their access to the site.

d. Limit User inputs

User inputs must be monitored as to the exact actions taken. Form inputs must be further modified to track the number of times a given user has tested the login and prevent further attempts by a given user. Active monitoring via the WAF should detect brute force attempts via a single user or botnet [F] and migrate the traffic and requests accordingly.

Further sanitisation of user inputs should be implemented in the backend php to identify potential malicious characters (for example ';' in a user input as an attempt to launch an SQL injection attack).

e. Database Management

The database needs further rules and protections added to keep the data secure. Backup scripts should be enacted to ensure the data is not lost. Sensitive data should be encrypted within the tables (such as passwords) and not stored in plain text.

5. Active Protection

Both prior and after the site is deployed, active monitoring and testing should be performed on the site to keep up with modern web security standards.

a. Dynamic Application Security Testing (DAST)

DAST programs are automated testing solutions to scan a given site in its entirety for potential vulnerabilities. Both passive and active reconnaissance are used to scan the site, and the found data is then used to simulate an attack.

i. ZAP

ZAP [3] is a popular DAST tool for complete automated scanning and attack simulation. This tool should be ran before any new deployments of the site are released.

b. **Active Penetration Tests**

The site should have regular penetration tests against it. It is recommended that this be at least annually [4], however all deployments should be actively tested prior to being pushed.

6. Appendix

▪ [A] Backend

Another term for the database and the code controlling it.

▪ [B] Principle of least privilege

A user either real or automated should have the least privileges possible whilst not restricting their required functionality. Nothing is given more privilege than required to complete the needed task.

▪ [C] Brute force attack

An attack in which the attacker attempts to guess the username and password for given users to escalate their privilege, or to steal information. The attack attempts many combinations of logins incrementally until a working one is found.

▪ [D] Root / Rooting

A common term within the hacking / pen test community meaning to have complete access to a target machine. It refers to the root level kernel of a Linux operating system.

▪ [E] Traffic manager

An application which sits as a proxy to a website. This distributes traffic correctly to each part of the site and can divert unwanted traffic away.

▪ [F] botnet

A large group of automated computer applications controlled via one host or group. They are normally malicious in nature.

References

- [1 OWASP® Foundation, "OWASP Top Ten," 2021. [Online]. Available:
] <https://owasp.org/www-project-top-ten/>. [Accessed 01 05 2021].
- [2 OWASP Foundation, "Source Code Analysis Tool," [Online]. Available:
] https://owasp.org/www-community/Source_Code_Analysis_Tools. [Accessed 01 05 2021].
- [3 OWASP ZAP, "OWASP Zed Attack Proxy (ZAP)," 2021. [Online]. Available:
] <https://www.zaproxy.org/>. [Accessed 02 05 2021].
- [4 M. Samarati, "How often should I schedule a penetration test?," IT Governance, 12 01 2017. [Online]. Available: <https://www.itgovernance.co.uk/blog/how-often-should-i-schedule-a-penetration-test#:~:text=IT%20Governance%20recommends%20having%20frequent,%2Dprofile%20or%20high%2Dvalue..> [Accessed 05 05 2021].

1910716 Database and application development - final

GRADEMARK REPORT

FINAL GRADE

79/100

GENERAL COMMENTS

Instructor

WEBSITE

Database structure and design 11/15 - Structure is OK, although quite simple. There is some normalisation on the role for the users table. You could have shown your skills further by having a few more columns on the cheese table eg country of origin, and this would also have allowed you to search on multiple fields (see below) and show more complex joins.

Range of PHP code demonstrated

28/40 - Search is OK - it could do with a wildcard adding (%) as at the moment an exact match is necessary eg "Cathedral City" - nothing comes up on "Cathedral". Search facilities are simple because the cheese table is simple, it would have been nice to give the user a choice of fields (eg find all French-origin cheeses) and therefore show more complex searches. Would have been nice to see a picture of the cheeses too! Similarly on the user search, only search by username is allowed, you could have used some other fields

Login is OK - it would have been nice to have some feedback when entering the wrong username/password, as it is, it goes to a blank screen with a few error notices. 2 states are managed.

Register is good with validation and prevention of duplicate usernames

Good use of paged results - important as the website grows. Be careful though - you have two on the page

and when you click on a page for one, it changes both. Use a different URL variable (something other than s) for one of them to avoid this.

Edit is OK - you might want to populate the fields with the current information to help the user.

There is a problem on Delete - whichever one I choose from the pull-down menu, it deletes the last one in the database. I think there's a bug in the code here.

Overall integration of website

13/15 - Layout is good overall, you might want to make the background colour of the menu a bit darker so that it stands out more, and maybe line things up on the left hand side. There are a couple of spelling mistakes, so get it proof-read before selling cheese live on the internet! :) A good effort overall though and I've definitely learnt something about cheese! ;)

REPORT

Security issues identified 13/15 - A good number of threats identified. Use some examples/code fragments to illustrate your answer where you can, to get the point across to the user.

Consideration of security issues for website 14/ 15 - You have shown a good insight here and made suitable recommendations with technical detail included. Very good.

TOTAL: 79%

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7
