Client Report

100287756 – Alfie Arundell-Coote
100287814 – George Beales
100301701 – James Wright
CMP-6045B

1 Introduction

At a basic level, a food blog website is intended for users to post their meals for other users to see. The complexity can range dramatically however with more extravagant features then in turn comes more chance for a security breach. With this website we have created a simple front end with just the essential features so that we can maximise the security potential.

We have two main objectives with this site: the first is making it easy to use so that any user of any level can utilise our site and secondly to ensure that the website is watertight and safe from a multitude of potential cyber attacks. In order to complete our first objective we have decided to utilise white space to our advantage by not crowding our site with unnecessary colours, images or buttons. For example our navigation bar only has 3 buttons and a search bar as we felt like if we added much more it could confuse some users.

Our second goal is achieved via implementing certain mitigation's in order to counter security vulnerabilities which are: account enumeration, session hijacking, SQL injection, Cross-site scripting and cross-site request forgery. We believe these to be the main 5 common vulnerabilities that could be abused in our site. These security mitigation's do not affect the usability, for example our cookies expire after one hour so that if the user quickly clicks off of the site, they can get back online without the needs to generate new cookies.

2 How ethics are essential to secure web development

Now when creating a secure website there are ethics that come into play. Having secure encryption is perfect so that people's data is safe and secure and not liable to theft from malicious sources. However a downside to this is that since everything is encrypted, if anything malicious is to take place on our site, it could be very difficult to actually find the malicious people in question. However we need to focus on the user as our product is designed for them and by protecting them, we believe this to be the ethical approach. We also have a system administrator present who can delete posts that may be inappropriate.

Ethics come into play in other scenarios such a data breach. Companies can be faced with the conundrum that if there is a breach, do they either tell the users immediately or do they try and mitigate and contain the problem before it spreads. We attempt to do both as we make our users sign up with an email address so in the event of a data leak we can alert them via the use of email.

3 What mitigation's were coded for each security vulnerability, and should include discussions of pre-built libraries, hashing, encryption, usability etc.

3.1 Account enumeration

Generic invalid login message that does not specify whether or not a username exists

HTTP response and the time taken to respond to incorrect password and the username being already taken is the same

We protected the registration process from account enumeration by implementing a captcha system in order to stop quick enumeration of usernames. (ie the search for taken usernames via botting cannot occur) also mention the fact that we have no way of telling the user that a username is already taken without exposing its existence as a whole. However we have figured that this is the best response as it does not impact the user too heavily.

3.2 Session hijacking

(ask james if cookies are encrypted)

We parse our sessions in cookies as we believe this solves a lot of the session hijacking possibilities

3.3 SQL Injection

We have countered against traditional string SQL injection as we use something called MongoDB which utilises a specific data format called BSON where queries are represented as binary data (ones and zeros) so direct injection through a string (text) is not available. no sql code at all, everything is done through a shcema, all sql statements are based on a model which stops a user from injecting sql as they have no access to the sql code whatsoever the way the database is accessed the sql the model cannot be queried

3.4 Cross-site scripting

We implemented a node library called helmet which allowed us to use a content security policy. This allowed us to specify where we want the browser to run scripts and also use some built in features of the helmet library. This stops a malicious user from being able to inject HTML into text that the server will serve to the user because the script will not run from places that where we did not specify it could not run in the first place.

3.5 Cross-site request forgery

Our get requests don't change any pre-existing data. Nothing yet EXPAND

captcha helps as the user needs to auth before confirming anything for each action all cookies we have implemented cannot be used in cross site request forgery we also have a secret cookie which retrieves a key from a .env file this stops users from all cookies are unique and secret meaning they are unpredictable, thus means that a user cannot create a dummy cookie and access things they are not intended to access

4 What Authentication method/s were coded, and evidence of how they increase both usability and security.

4.1 2FA

Hackers do not care if a business is big or small, if it is liable to an easy attack they WILL go for it, this is why we have decided to utilise 2FA (2 factor authentication) when logging in to our site. 2FA halts an attack via the use of bots which essentially rapidly guess passwords against an account as once the password is guessed correctly the user will receive a 5 character code that they then need to input into the site so they can log in. This means that if a hacker would like to gain access to a persons account on our site, they need to also have access to said persons

email address which is significantly harder to obtain than just a username and password for our site.

4.2 Passwords

Passwords are the base line of defense against unauthorised access to an account. The user can pick their own password, ideally something memorable to them but not anything too specific such as their name or just "password". The stronger the password the more protected the account and therefore the less are the chances for it to be breached.

4.2.1 Hashing our Passwords

We have irreversible hashing that means that with our B-crypt library that if the same word was hashed for a second time it would never be the same. the way we authenticate passwords is we compare hashes to one another and we also use 8 salts which encrypt the password further.

4.3 Captcha

We have lastly implemented captchas in another bid to halt brute force attacks on our users. Having a captcha present on login prevents the use of bots to rapidly attempt to log in to our site as I mentioned earlier in the 2FA section as it requires the user to verify a number of images before logging in. The second place we have implemented a captcha is in registration, this prevents the use of bots to create a multitude of accounts on our site which could lead to many fake accounts being created and in turn wasting resources.

5 Discussion and evidence of user testing

when a post is deleted, the ID of the post gets added to the URL. If the URL was changed before the URL gets checked, the cookie/session is checked to see if the person deleting it has actually made this request.

6 Discussion and evidence of system unit testing

we are using a library called jest which tests individual functions in the route folder.

7 Conclusion

if we had more time... more testing coverage ie more code gets tested at each point

References

A Appendix A

B Appendix B