Software Engineering 2: Developing a Secure Web Application

A. Kearns, M. Williams, S. Gibson, C. Irvine

1 Introduction

A local music venue (the University of East Anglia's LCR) is looking to update its website and provide an interactive portal so that users can log in and discuss their experiences as well as upcoming events. Meeting these client requirements will require the design and implementation of a secure web application that is protected against a wide variety of attack vectors including but not limited to: SQL injection, cross-site scripting, cross-site request forgery, and account enumeration. This document serves to give an overview of the design and implementation decisions made in the process of creating the web application, including a complete breakdown of every security threat and mitigation strategy considered during the course of development.

2 Choice of Technology

A number of web technologies exist that can be used for the development of an interactive portal, allowing users to log in and discuss things. Detailed below are the decisions and justifications made for the technologies used in the implementation of the web application.

2.1 PHP & Laravel

Laravel [1] is an open-source PHP web framework allowing users to develop web applications that follow the model-view-controller (MVC) pattern. Its features include a routing engine, authentication, configuration management, and many more. Its main design focus is modularity, with built-in and 3rd party libraries integrated with Composer [2], a PHP dependency manager. It also includes a rich set of security features such as protection against SQL injection via prepared statements, forcing HTTPS, escaping HTML and content to prevent against cross-site scripting, and protection against cross-site request forgery via tokens.

It is for these features and the fact that two members of the team were already experienced in using PHP in combination with Laravel to develop and deliver web applications that Laravel was chosen as the project's web framework.

2.2 MySQL

MySQL [3] is an open-source relational database management system (RDBMS) developed by Oracle Corporation. It is very widely used and has an extensive support network, and is also one of the four RDBMS solutions supported by stock Laravel.

2.3 NGINX

NGINX [4] is an open-source web server and load balancer. As of December 2016 it is the 2nd most used web server with 38.2% of active websites using it [5]. NGINX was used as the web server to host the application, showing that *Music Forum* provided server security mitigations, as well as traditional application security.

3 Design

The system design was mostly focused around developing a secure online forum that is resilient to any kind of security vulnerability. Taking into account the client brief, requirements can be identified:

- Users can create accounts and log in to the system.
- Accounts can have privilege levels to allow moderators and administrators to maintain the forum.
- Users can create threads to discuss their experiences as well as upcoming events.
- Users can comment on threads to continue the discussion.
- Users can reset their password if they forget it.
- Users can view the profile pages of other users.
- The system is not vulnerable to any kind of security risk.

These requirements were used in the implementation stage to inform decisions regarding project scope, as well as being used to evaluate the overall success of the project in acceptance testing.

3.1 Entity-Relationship Diagram

The entity-relationship diagram (seen in Figure 1) shows the relationships between entities in the database, which could potentially expose any weaknesses in the database design or security vulnerabilities in terms of database design.

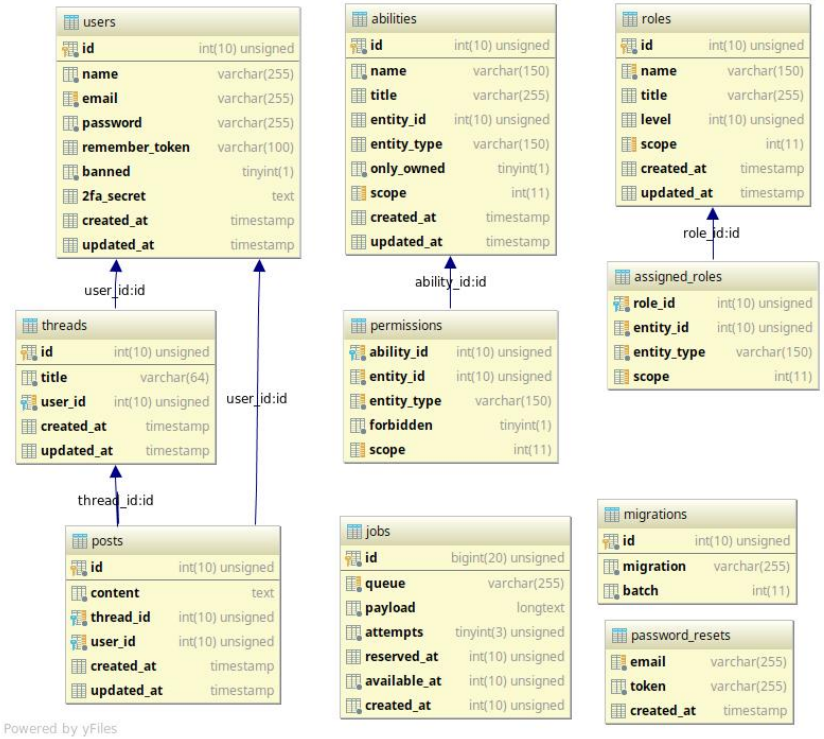

Figure 1: *Music Forum* Database entity-relationship diagram

3.2 Gantt Chart

Shown in Figure 2 is the Gantt chart created to manage how long each task in the implementation should take.

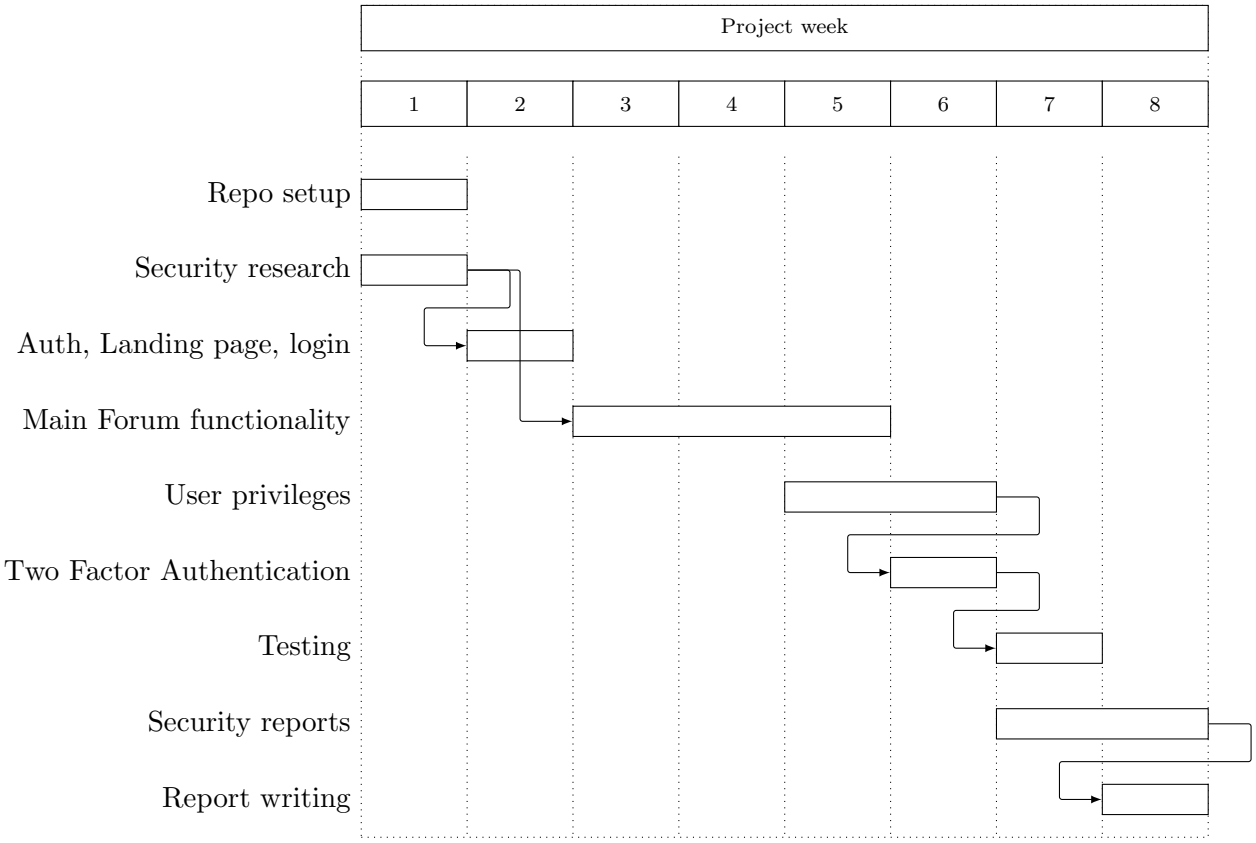

Figure 2: Gantt chart for web application development

4 Implementation

In this section the intended uses of *Music Forum* (see Section 4.1) will be explored in addition to the various security mitigations that have been included throughout development (see Section 4.2).

4.1 Usage

The expected uses of the *Music Forum* web-app is that attendees of the UEA LCR venue will share and discuss their experiences of gigs and events at the LCR. A user of the site will register an account, opting in to use 2 Factor Authentication (see Section 4.2.7) if the user wishes. As part of the registration process the user must create a password, the chosen password will then be validated (see Section 4.2.8), upon validation the account will be created and the user will be asked to login.

Once logged in the user can create a thread and make posts on that thread or a thread belonging to a different user. User's may also edit or delete posts belonging to them. Users are also to view their account, and the accounts of others. When viewing an account, users can see the name, join date, number of threads and posts and a list of the users current threads and posts. When viewing their own account, users can enable or disable 2 Factor Authentication. Finally, users are able to log out of *Music Forum*.

4.2 Security Mitigations

The principal focus of the initial development of *Music Forum* was to create a simple secure forum for the LCR venue. As a result, we have attempted to cover the most prevalent security vulnerabilities to both applications in general and forum websites. The Open Web Application Security Project (OWASP) [6] produces a top 10 list of the most critical security risks facing web-apps, as well as suggesting methods to prevent them [7]. OWASP Zap is a free tool developed by OWASP which will scan a web-app for any security risks [8]. The development of *Music Forum* was initially completed with the use of both the top 10 OWASP list and the use of the OWASP Zap tool. Further mitigations were implemented as a result of testing (see Section 5).

4.2.1 SQL Injection

SQL injection is an attack used to gain access to or mutate data within a database that the user would not normally have access to. Malicious SQL statements would be typed into a form field and submitted, where an unprotected system would execute these statements, potentially crippling a database. *Music Forum* is protected against an SQL injection attack through the use of 'prepared statements' – whereby a query is created with any parameters left blank, then compiled and stored by the database without executing it. Finally, when the query is ready to be executed, parameters are 'bound' to the query and the database executes it, possibly returning a result. This protects against SQL injection because the parameters supplied later are not compiled by the database, they are only substituted into it as data.

```
$id = $request->thread_id;
$thread = DB::select(DB::raw(
    "SELECT *
    FROM threads
    WHERE id = :id"
), [$id]);
$thread = new Thread($thread);
```

```
DB::insert(
    "insert into posts
    (content, thread_id, user_id)
    values (?, ?, ?)",
    [$request->content, $request->thread_id, $user->id]
);
```

Figure 3: Example database queries using prepared statements.

4.2.2 Cross-Site Scripting (XSS)

XSS is an attack that causes malicious client-side scripts to be executed on web pages viewed by other users. It occurs when any part of a website that accepts user input does not sanitise the input. For example, if the search bar of a website does not sanitise its input, then a user could enter arbitrary JavaScript within a `<script>` tag which would get executed. If this input is stored in a URL query string then a link can be crafted and sent to an unsuspecting user and cause arbitrary code to be executed by their browser. An even more malicious variant of XSS is when unsanitised input is stored in a database and used somewhere in the site, causing it to be executed every time someone visits that part of the website. *Music Forum* is protected against XSS attacks using Laravel's Blade templating engine, which can sanitise and escape input and remove the possibility of any valid HTML getting submitted.

4.2.3 Cross-Site Request Forgery (CSRF)

CSRF attacks force end users to execute unwanted actions through a web-app that they are currently authenticated on. These attacks specifically target state-changing requests, and usually are run alongside social engineering operations (for example sending a link via email). When a successful CSRF attack is triggered there is often no indication of this to the victim. The prize of CSRF attacks is usually the transfer of funds or acquirement of email addresses.

These attacks can take one of two forms, `GET` or `POST`, which form the attack takes depends on the functionality of the application the attack is taking place on. `GET` attacks are based around mimicking a `GET request` from the application that will execute a malefic function, such as transferring funds to the attackers account. `POST` attacks are identical to `GET` attacks with the exception that they are triggered by submitting a malicious form.

In either case CSRF attacks are prevented through the use synchronised tokens, where a large random value (token) is generated for every session of the app (shown in Figure 4. Each user operates within a session and the requests must include the token that belongs to the user. Laravel will automatically generate this token when using the `@CSRF` Blade derivative. Other mitigations are verifying the origins of the requests and the use of reCAPTCHA technology.

Figure 4: An example server request that includes the session token.

4.2.4 Account Enumeration

A web-app is vulnerable to Account Enumeration when presented with incorrect login credentials, the system informs the attacker if it is the account identifier or password that is incorrect. In conjunction with limitless login attempts, this can allow a determined attacker to guess the password to an account (either manually or through a dictionary attack). The harm caused by Account Enumeration systems can extend to just the knowledge that an email belonging to an individual has an account on the web-app.

To mitigate Account Enumeration *Music Forum* does not declare whether a user has provided an incorrect email or password when attempting to login. Additionally users must verify they are not a robot via the reCAPTCHA system (see Section 4.2.6), and once they have exceeded 10 attempts to logging in they will be forced to wait 5 minutes. This forces attackers to manually spend a lot of time accessing a users account as *Music Forum* gives them no information as to how close they are to gaining that access. Users who have had passwords leaked in the past are also protected on *Music Forum* through the use of Leaked Password Verification (see Section 4.2.8).

To further protect against Account Enumeration, and more specifically timing attacks, queues were used when resetting passwords. A timing attack is when an attacker can determine whether a user has an account based on the time it takes reset password request to be carried out. This attack was mitigated by queuing each password reset as a job, which results in each password reset request taking the same time.

4.2.5 Access Control

Access Control attacks are carried out to circumvent or bypass access control methods (such as logging into user accounts) in order to either steal data or user credentials. The latter is particularly dangerous for users who use the same password on multiple web-apps.

These attacks can either be *aggregation* or *spoofing* attacks. Aggregation attacks are performed by collecting several pieces of insensitive information in order to draw conclusions that form sensitive information. Commonly these can include password, dictionary, brute-force, birthday and sniffer attacks. Spoofing attacks attempt to imitate a trusted user, fooling security systems, allowing attackers to alter information or lure the system to send them confidential data. Examples of these are email, phone, social, phishing (wide range of targets) and spear phishing (specific targets chosen) attacks [9].

Music Forum is protected from Access Control attacks by allowing users to use Two Factor Authentication (see Section 4.2.7), the use of reCAPTCHA (see Section 4.2.6) and enforcing a strong password policy, including the vilification of leaked passwords (see Section 4.2.8).

The control that users can exert over *Music Forum* is limited to only the sections of the app that is directly owned by their user account. This includes the account itself and the posts that the account makes. The second aspect of Access Control is the level of access that each user, including administrators, have over the *Music Forum* systems. *Music Forum* limits this access for standard users to only their own account whereas the administrative accounts can take control of other users posts and threads in order to maintain the site.

4.2.6 reCAPTCHA

reCAPTCHA is part of the Google SafetyNet API, which provides simple tools to prevent common malicious attacks that are usually carried out by bots. Powered by an advanced risk analysis engine, reCAPTCHA prevents bots from logging into *Music Forum* whilst allowing the valid users to log in without issue. Within the *Music Forum* security system, reCAPTCHA defends against a multitude of attacks including CSRF (see Section 4.2.3), Account Enumeration (see Section 4.2.4), Access Control attacks (see Section 4.2.5) to name but a few.

reCAPTCHA also improves the quality of life for *Music Forum* users; due to the lack of bots using the forums, users will not be subject to unwanted and potentially malicious advertisements that infest other forum web-apps.

```html
<div class="form-label-group">
    {!! NoCaptcha::display() !!}
</div>
```

(a) HTML code snippet

```php
/**
 * Get a validator for an incoming registration request.
 *
 * @param  array  $data
 * @return \Illuminate\Contracts\Validation\Validator
 */
protected function validator(array $data)
{
    $rules = User::rulesForCreating();

    // Can only do google recaptcha in production.
    if (config('app.env') == 'production') {
        $rules = array_merge($rules, ['g-recaptcha-response' => 'required|captcha']);
    }

    return Validator::make($data, $rules, User::messages());
}
```

(b) PHP code snippet

Figure 5: HTML and PHP code snippets that enable the reCAPTCHA tool on *Music Forum*.

The implementation of reCAPTCHA is simple, once the reCAPTCHA API key was included in the environment file we only had to include reCAPTCHA box in the HTML (see Figure 5a) and initiate the check on the login and registration routes (see Figure 5b).

4.2.7 Two Factor Authentication

Whilst *Music Forum* has a basic identification/password authentication system to protect user data, that is safe from Account Enumeration (see Section 4.2.4) and validates leaked passwords (see Section 4.2.8), there are further mitigations that can be made. To further protect accounts,

a user can opt in to use the Two Factor Authentication service of their choice, *Music Forum* recommends *Google Authenticator*.

A user can enable or disable two factor authentication on their account. A user with this enabled has a secret key associated with their account, this is used by Google to determine if the code given by the user on login is valid.

4.2.8 Leaked Password Validation

In the last few decades, there has been a series of hacks that release sensitive information – including passwords – into the public domain. Some users may not be aware that their password might have been compromised. As a result when a user creates an account on *Music Forum*, the server will search the proposed password into `https://haveibeenpwned.com/passwords` [10]. If that password has been leaked more than 100 times then the password will be rejected by the *Music Forum* server with the user being informed that their chosen password has been compromised. An industry standard hashing algorithm was used to encrypt passwords before storing them in the database, to ensure even if the database is breached, the passwords are not distinguishable.

4.2.9 Header Configuration

Web application headers (also known as HTTP headers) dictate what scripts are able to run on the server, they can also be used to pass information from the client to the server through the use of *request* headers. The *Music Forum* server is set up so that only "safe scripts" are executed when loading *Music Forum* pages. This is done using the `HTTP Content-Security-Policy` (CSP) directives [11]. Currently, the *Music Forum* server will only run scripts that the developers have directly included in the development and *white listed* sources - such as bootstrap. These policies extend to style sheets, fonts, images and other media.

4.2.10 Version Control

GitHub [12] was used throughout the development of *Music Forum* to provide version control as well as critical security features. New features of *Music Forum* are developed within their own branches on GitHub. This means that the deployed version of *Music Forum* are not affected by incomplete features, ensuring the server and the database remain safe. When a feature is finished, it must be reviewed and tested by the other developers (to ensure system integrity) before it is merged with the deployed version. Additionally, GitHub will alert the developers when a package is a security risk, so that the issue can be rectified (shown in Figure 6.

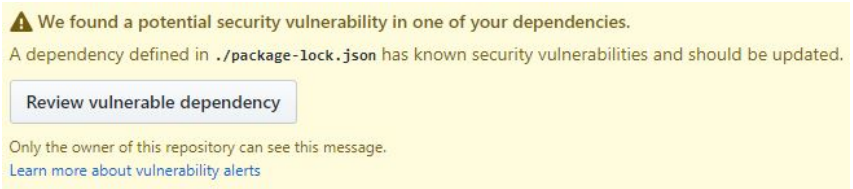

Figure 6: An example vulnerable dependency alert.

4.2.11 Bugsnag

During the development of *Music Forum*, an error monitoring platform (Bugsnag) was used in order to ensure the integrity of *Music Forum* beyond what the IDE and tests could detect. When

there are errors in the source code of a system, it opens up even the most secure components to attack. It is one of the OWASP top 10 risks to web apps [7].

Bugsnag [13] will monitor the current live build of *Music Forum*. When an error occurs during the operations of *Music Forum*, the system administrators will receive a notification from Bugsnag (shown in Figure 7). Whilst *Music Forum* has over 50% test coverage, it is not perfect and errors can and will still occur. By including Bugsnag in the *Music Forum* pipeline, the developers can focus on new features and security implementations without manually checking for errors. No application is completely secure, so documenting any breaking errors is essential in order to provide quick fixes.

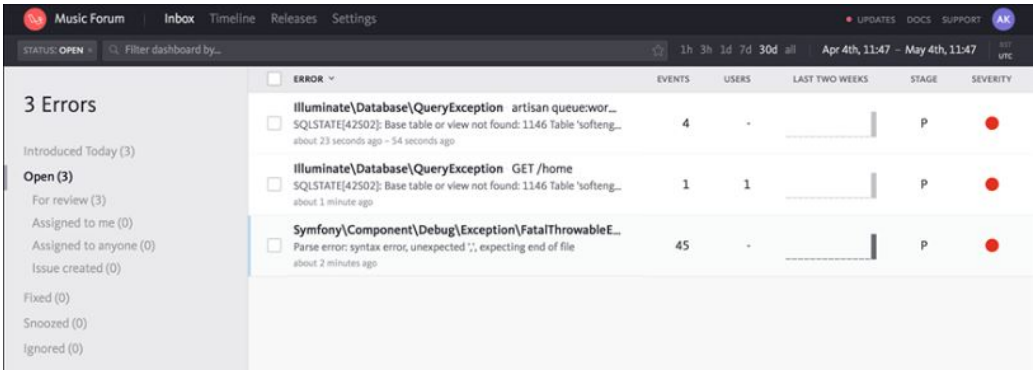

Figure 7: The Bugsnag dashboard that displays the most recent errors and informs administrators. Please note these errors were implemented on purpose to provide an example of the Bugsnag dashboard.

4.2.12 Server Configuration

The *Music Forum* server is a critical component of the system that must be protected. It manages the database which stores all the user information and credentials, as well as the source code for *Music Forum* itself. If an attacker gains control of the server the entire system would be vulnerable, and susceptible for further attack.

As a result the server can only be accessed with SSH (Secure Shell) authentication, which is recognised as one of the most secure forms of authentication available. The MySQL database has been initialised with the secure command which removes any bad error logging and removes the default users. Consequentially, only users that pass through the *Music Forum* authentication system (see Sections 4.2.4, 4.2.5, 4.2.7, and 4.2.8) can access a portion of the database. Finally, as described in section 4.2.9, the *Music Forum* domain is secured with HTTPS. Passwords that are sent from the client to the server cannot be intercepted by any attacker.

5 Testing

Music Forum was tested using a number of different methods to ensure it was secure against a multitude of attack vectors as well as functioning correctly.

5.1 Unit Testing

Music Forum was unit tested using PHPUnit [14]. Laravel includes support for unit testing with PHPUnit 'out of the box', and provides the necessary tools for developers to begin unit testing as soon as the application is set up. *Music Forum* has unit tests for all of its features to ensure that they are working as designed.

```php
/** @test */
public function can_access_when_authenticated()
{
    $user = factory(User::class)->create();
    $response = $this->actingAs($user)
        ->get('/home')
        ->assertStatus(200);
}
```

```php
/** @test */
public function cannot_access_when_not_authenticated()
{
    $response = $this->get('/home')
        ->assertStatus(302);
}
```

Figure 8: Example authentication unit test.

5.2 Header Testing

Music Forum was header tested using the service 'Security Headers' [15]. 'Security Headers' checks your web application to ensure that it issues the correct HTTP response headers, and gives a score from A+ to F based on how many were correct. According to their website this 'indicates a high level of commitment to improving security for your visitors'. As shown by Figure 9, *Music Forum* scored A+ on this test, which is the highest possible grade.

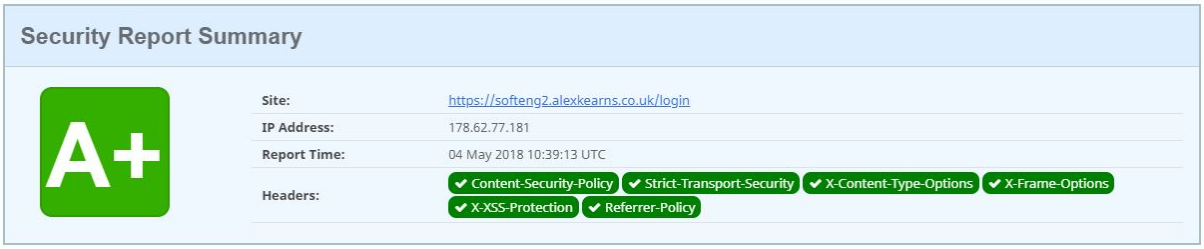

Figure 9: SecurityHeaders.io header rating.

5.3 SSL Testing

Music Forum used SSL Labs 'Server Test' [16] to examine its server setup and try to find out how effectively it is configured to use SSL. It examines the server certificate and looks into things such as protocol, key exchange, and cipher support. *Music Forum* scored an A+ on this test, which is the highest possible grade.

5.4 OpenVas

Music Forum used OpenVAS [17] – an open-source automated vulnerability scanner and manager – to scan for vulnerabilities. It is a large framework of a number of services and tools that offers a comprehensive vulnerability scanning solution. OpenVAS generates a report of any potential vulnerabilities that it was able to find after scanning has finished, and when it scanned *Music Forum* it did not find any major vulnerabilities.

5.5 OWASP Zed Attack Proxy (ZAP)

Music Forum used OWASP ZAP [18] as an additional automated security vulnerability scanner. It functions in much the same way as OpenVAS in that it scans a web application for vulnerabilities and generates a report detailing any findings afterwards. Much like OpenVAS, the report generated for *Music Forum* did not detail any major vulnerabilities which indicates that *Music Forum* is secure against a wide variety of common attack vectors.

References

[1] Taylor Otwell. Laravel - the php framework for web artisans. https://laravel.com/, April 2018. (Accessed on 04/28/2018).

[2] Nils Adermann and Jordi Boggiano. Composer. https://getcomposer.org/, April 2018. (Accessed on 04/28/2018).

[3] Oracle Corporation. Mysql. https://www.mysql.com/, April 2018. (Accessed on 04/28/2018).

[4] NGINX Inc. Nginx — high performance load balancer, web server, & reverse proxy. https://www.nginx.com/, April 2018. (Accessed on 04/28/2018).

[5] W3Techs Web Technology Surveys. Usage survey of web servers broken down by ranking. https://w3techs.com/technologies/cross/web_server/ranking, December 2016. (Accessed on 04/28/2018).

[6] Open Web Application Security Project. Open web application security project. https://www.owasp.org/index.php/Main_Page, January 2018. (Accessed on 04/29/2018).

[7] Open Web Application Security Project. Owasp top 10 - 2017. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf, October 2018. (Accessed on 04/29/2018).

[8] Open Web Application Security Project. Owasp zap. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project, April 2017. (Accessed on 04/29/2018).

[9] Ryan Fahey and Certified Information System Security Professional. Cissp prep: Mitigating access control attacks. http://resources.infosecinstitute.com/category/certifications-training/cissp/domains/identity-and-access-management/mitigating-access-control-attacks/#gref, 2018. (Accessed on 04/29/2018).

[10] Troy Hunt. Have i been pwned. https://haveibeenpwned.com/Passwords, 2018. (Accessed on 04/30/2018).

[11] MDN Web Docs. Http headers. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers, 2018. (Accessed on 05/02/2018).

[12] GitHub, Inc. Github. https://github.com, 2018. (Accessed on 05/02/2018).

[13] Bugsnag, Inc. Bugsnag. https://www.bugsnag.com, 2018. (Accessed on 05/02/2018).

[14] Sebastian Bergmann. Phpunit the php testing framework. https://phpunit.de/, April 2018. (Accessed on 04/29/2018).

[15] Scott Helme. Analyse your http response headers. https://securityheaders.com/, April 2018. (Accessed on 04/29/2018).

[16] Qualys Inc. Qualys ssl labs. https://www.ssllabs.com/, April 2018. (Accessed on 04/29/2018).

[17] Greenbone Networks. Openvas. http://www.openvas.org/, April 2018. (Accessed on 04/29/2018).

[18] Open Web Application Security Project. Owasp zed attack proxy project - owasp. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project, April 2018. (Accessed on 04/29/2018).