A Comparison of Modern Backend Frameworks Protections against Common Web Vulnerabilities

Jan-Marius Grünwaldt
Tufts University

Fall 2019

Abstract

The web evolved to become one of the most important platforms for applications today. Given the public availability of those applications, it is very easy to run attacks against web servers. The vulnerabilities of web servers, regarding those attacks, can be classified into different categories. A few of the most common vulnerabilities include Injection, Broken Authentication, Sensitive Data Exposure, and Cross-Site Scripting (XSS) flaws. That means we can expect, ahead of time, that our application needs to include protections against certain kinds of attacks. It, therefore, seems reasonable to let the need for protections influence the design of the tools used to build our application. One of the main tools used during the development process of a modern web application is the backend framework. In this paper, we compare the modern backend frameworks Django, Spring, Laravel, and Express.js from a security standpoint. We evaluate each framework by looking at a variety of security issues and assessing, what protections the framework offers against them. In this context, a secure framework is one, that makes it as easy as possible to develop applications with protections against the most common security vulnerabilities. Even though we will focus on security issues created during the development of an application, we will also address security flaws arising from the use of the framework itself.

1 Introduction

To make this comparison as interesting and relevant as possible, we are going to choose only widely-used, free and open-source frameworks that are all written in different underlying languages. The first framework we are going to look at is Django, a Python Web framework, which claims that it "helps developers avoid many common security mistakes, such as SQL injection [and] cross-site scripting [...]"[1]. Spring is a generic open-source Java framework that can be used in many ways besides web development and through the use of Spring Boot claims to have "[s]implified security"[2]. Laravel is a PHP web framework which "makes your applications more secure by default"[3]. Express.js is a backend framework that describes itself as a "[f]ast, unopinionated,

[1]Source: https://www.djangoproject.com/start/overview/
[2]Source: https://spring.io/
[3]Source: https://auth0.com/blog/why-laravel-is-the-recommended-framework-for-secure-mission-critical-applications/

minimalist web framework for Node.js"[4]. The protections against vulnerabilities we are going to look at are greatly shaped by the OWASP Top 10 List of Web Application Security Risks[5]. We will be looking at 4 of the most important security aspects, namely Injection, Authentication, Sensitive Data, and Cross-Site Scripting. With that being said, one also has to mention that many OWASP TOP 10 issues can not be mitigated by the framework at all. Some issues are controlled by configuration, web server or the concrete instance of the application and need specific solutions that can not be abstracted to the framework level. Examples include shipping or deploying applications with easy default credentials, enabling web server directory listing, or storing sensitive data unnecessarily. For each of the 4 mentioned security aspects that are influenced by framework choice, we will be looking at the protective measures that can be applied to them. In the end, we want to find out the differences between common web backend frameworks when it comes to security.

2 To the Community

Frameworks play a pivotal role in the security of an application. If every developer had to re-implement basic security features such as authentication for every application, this would always include the possibility of security vulnerabilities being introduced. In that way, because frameworks can provide a lot of functionality already, such as authentication, database access and more, they can greatly help developers, who now do not have to develop those features themselves. At the same time, web security is comprised of many different points outside of authentication. Developers have to deal with brute-force attacks, cross-site scripting, injection, and the protection of sensitive data. In the following, we will identify some of the most important protections web applications need to include, to not leave their applications vulnerable. Based on this we will formulate questions that can be used to evaluate frameworks. We will then also discuss those questions for each framework. In the best case, developers can rely on the expertise of the developers of the framework, who spend a lot of time getting security and features right. The question now is, in what way do the protections offered by frameworks differ? What difference does the framework choice make for developers? If developers are aware of the advantages, or disadvantages regarding the security of certain frameworks, they can make a more informed choice. In the end, they might build an application that is better equipped to handle attacks, and better protects the confidentiality and integrity of their data and the application's availability.

[4]Source: https://expressjs.com/

[5]See: OWASP Top 10 - 2017, The Ten Most Critical Web Application Security Risks, https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

3 Background

3.1 Injection

Injection flaws can occur whenever an application sends user-supplied data to an interpreter, and that data is not properly escaped. For this project we are going to focus on SQL Injections as one of the most common example. A protection against that is the use of prepared statements, that clearly define what parts of the queries are supplied parameters, and therefore allow for proper escaping. Another defense is the use of Object Relational Mapping (ORM) tools, which allow the developer to perform CRUD operations on the database, without having to work with queries directly. They therefore auto-escape supplied parameters and protect from injections where the attacker can execute arbitrary SQL commands, by terminating the SQL statement with a quote and semicolon. Therefore, when analyzing frameworks we want to answer the following questions:

- Does the framework use parameterized queries by default?
- Does it provide Object Relational Mapping Tools (ORMs) to avoid having to work directly with database queries for CRUD-Operations?

3.2 Sensitive Data Exposure

Sensitive Data Exposure occurs whenever sensitive data is inadequately protected, i.e. can be accessed without proper authorization. Therefore, we are interested in the authentication system the framework offers. Another closely related important issue is the hashing of the user's passwords in the database. Passwords are hashed to hide the plain text passwords of users in the case of unauthorized database access. Many hashing algorithms are not suitable for password hashing such as MD5, and even SHA-512, which simply take too little time to calculate on fast hardware. Also, a salt must be used during the hashing, to make the use of pre-computed hashes ineffective. A salt is a random string that is added to the password during hashing. As a result of this, even hashing the same password twice will create distinct hashes. Therefore, for sensitive data exposure we want to answer the following questions:

- What kind of authentication system does it have?
- Does it use a modern cryptographic hashing algorithm with a salt by default?
- Can you use modern hashing algorithms, such as Argon2, winner of the Password Hashing Competition[6]?

3.3 Authentication

Because web applications are usually set up in a way that makes them available to the public, they must be protected against numerous attacks that try to bypass the authentication system.

[6]See: https://password-hashing.net/

A common attack to gain unauthorized access to systems is using some variation of a brute-force attack. They all come down to an attacker making a large number of login attempts, to guess a correct combination and gain access to the system. It is therefore important that web applications deploy mechanisms to limit the login attempt rate, to make brute-force attacks ineffective. Another important factor is the password a user sets for his account. Since there exist lists of the most commonly used passwords, it makes sense to not allow the use of those. Even if the login attempt rate is limited, very easy password ("123456") can be guessed in a small number of attempts by attackers. Additionally, a complexity requirement is important, since every character of additional length increases the time required to crack the hash by a huge factor. If the users password is breached, using a long and random password is important, since the plain text password can be generated from the hash by brute-force attacks. User sessions should furthermore have a timeout, and failed login attempts should be logged to be aware of possible currently running attacks. Therefore, for authentication we want to answer the following questions:

- Does it provide means to defend against brute-force attacks?
- Does it provide means to avoid common passwords and enforce password complexity?
- Does logging in users have a timeout by default?
- Does it log failed login attempts?

3.4 Cross Site Scripting

Cross site scripting is a type of attack where malicious code is injected into a trusted website. It is only possible when the website is displaying user input. For example, instead of sending a comment to a video, the attacker sends executable code to the server. If the input the code is included in the correct HTML context, the victim's web browser will treat it like it was originally included on the web page and execute it. An important protection against cross site scripting is escaping variables that are delivered to the front-end by default. So if the user input is stored in a variable and displayed, it won't be able to open a 'script'-tag, and the code will be treated as plain text. Another important factor in cross-site scripting is the different page contexts, that all require different escaping. For example, if the user input is included in the HTML body, it is enough to escape special HTML characters, such as < and >, and the attacker can not inject a 'script'-tag. On the other hand, if the user input is included in attributes, or inside a script-tag, it requires different escaping. Therefore, for cross site scripting we want to answer the following questions:

- Does the framework automatically escape content delivered to the frontend?
- Does it provide escaping based on page context (body, attribute, JavaScript, CSS, or URL)?

4 Action items

4.1 Django

Injection

Django provides Object-relational mapping out-of-the-box, and is recommending developers to use this as their default way to communicate with databases.[7] It also uses prepared statements by default, when actually executing raw queries, even though the Django Documentation is not recommending the use of raw queries.

Sensitive Data Exposure

Django comes with an user authentication system. It handles user accounts, groups and permissions. By default Django uses the PBKDF2 algorithm with a SHA256 hash[8], which is recomnened by NIST. It should be sufficient for most applications. Argon2 can be installed easily, but requires a third-party library.

Authentication

Django does not include brute force protection by default, but the package *django-axes* is a widely-used package enabling access rate limits. Also, failed login attempts are not logged by default, but a $user\_login\_failed$ event is fired and can be equipped with an handler, and *django-axes* is capable of logging failed attempts, too. When setting up a new Django Project with the *startproject* command, there are a nubmer of password strength requirements set by default, such as a minimum length of 8 characters. This can be further customized via $AUTH\_PASSWORD\_VALIDATORS$, which also enables developers to use the *CommonPasswordValidator* to ban the use of too common passwords. The default session timeout is two weeks. The developer has to manually change this, if he wants to either require login again after closing the browser or a smaller session duration. This is definitely too long.

Cross Site Scripting

Django escapes strings passed through a template variable by default. The developer has to explicitly mark strings that contain HTML and should be rendered. Django does not provide context-based escaping, such as inside script-tags, HTML attributes, or CSS.

4.2 Spring

Injection

If you use Spring with a library such as Spring JDBC, it will be using prepared statements by default. Spring also provides support for ORMs, through libraries such as Hibernate. It is not

[7] The Django Documentation recomends developers to explore ORMs before writing their own raw queries here: https://docs.djangoproject.com/en/3.0/topics/db/sql/#executing-custom-sql-directly

[8] See: https://docs.djangoproject.com/en/2.2/topics/auth/passwords/

directly integrated into the framework though, since Spring is not a web-only framework, but a more general framework.

Sensitive Data Exposure

Spring does not come with a user authentication system out of the box and needs external libraries for it. They require proper integration and setup, making Spring in this regard more complicated than Django/Laravel, truly web-only frameworks. Spring does not even have a default way to store user details and passwords. This means the developer is responsible to set up password hashing. The Spring Interface PasswordEncoder provides the following hash functions: MessageDigestPasswordEncoder implements MD5 and SHA-512, Pbkdf2PasswordEncoder implements PBKDF2, BCryptPasswordEncoder implements BCrypt, and SCryptPasswordEncoder implements SCrypt. MD5 and SHA-512 should never be used to hash passwords. Modern hardware can calculate a lot of hashes really fast.[9] What is required is a hashing method that includes a work factor, such as PBKDF2. This is a problem for Spring, since the authentication requires more set-up and configuration than Django, and might cause data exposure vulnerabilities here by using insecure hash functions.

Authentication

Spring does not provide brute-force protection out of the box. There is currently no library to implement this feature either, which is problematic. Similarly, in order to log failed attempts with Spring, developers have to build this themselves, including both the event and event listener. Spring also does not have a default password complexity set up. This requires more work and attention to detail from the deverlops than using other frameworks. Spring also requires additional libraries for Session Management - for example HttpSession with Redis. It has a default timeout of 30 mins.

Cross Site Scripting

Spring provides a defaultHtmlEscape setting inside its web.xml config file, which enables autoescaping HTML generated from JSP form tags. Spring does not provide context-based escaping.

4.3 Laravel

Injection

Laravel includes support for SQL, and prepared statements are supported as well. Object-relational mapping is included in the framework, too.

Sensitive Data Exposure

Laravel includes authentication and claims to have almost everything configured by default[10]. This is a clear advantage over Spring. Laravel uses Bcrypt by default. This is good, since BCrypt is

[9] See: https://rietta.com/blog/bcrypt-not-sha-for-passwords/
[10] See: https://laravel.com/docs/5.0/authentication

specifically designed to store passwords. Laravel also provides support for argon2 without the use of external libraries. It does not provide support for other hashing algorithms by default.

Authentication

Laravel's default *LoginController* class uses a ThrottlesLogins trait by default, with a timeout being of one minute being applied after several failed login attempts. Laravel provides a *Auth\Events\Failed* event which can be equipped with an event handler to log failed attempts. By default, there is no event handler equipped, but the setup does not require external libraries. Laravel has an 8 character minimum password length, but has no other complexity requirements. This is problematic, because an 8 character numeric password is still easy to crack. Furthermore, Laravel provides no out-of-the-box way to ensure password complexity. Laravel comes with a rules validator for inputs, that uses regex expressions to validate inputs. This can be used to implement password validation, but requires manual set up. Laravel uses the PHP default session timeout, which is two hours.

Cross Site Scripting

Laravel comes with a template engine called Blade, that provides template tags, with autoescaping of HTML special characters. Blade does not provide escaping based on page context, but *escape* is an extension to do this.[11]

4.4 Express

Injection

To use databases with Express, external libraries are required. The library for MySQL support (*mysql*) provides support for prepared statements. It also requires *Sequelize* or other libraries for Object Relational Mapping.

Sensitive Data Exposure

Packages such as passport.js can provide user authentication management. Express also requires libraries for hashing, such as *bcrypt* or *argon2-ffi* to provide support for those hashing algorithms.

Authentication

Express does not come with configured brute force protection, but packages such as *express-brute* can be used. *Express-brute* also provides a *failCallback* that can be used to log failed attempts manually. A password complexity requirement has to be either implemented manually, or through a package like *express-validator* or *owasp-password-strength-test*. Using *express-session* package, there is no session timeout by default. However, modern browsers treat this like like a non-persistent cookie and delete it on exit. This still might be problematic for old browsers.

[11] See: https://github.com/laravelgems/escape

Cross Site Scripting

Express requires a template-engine (e.g. *pug*) to display template content. If this is installed, it provides similar protection against XSS like Laravel or Django. Also, there are packages like *express-sanitize-escape*, that can escape directly in code. There is no default way to escape based on context.

5 Conclusion

Prepared statements and ORMs are by now a standard procedure to protect against Injection vulnerabilities, and therefore there all frameworks offer some way to use those two. What differs, however, is the ease of integrating those tools into the framework. Django and Laravel support both out of the box, Spring and Express.js require the installation of additional packages for it.

Concerning Sensitive Data Exposure, Django and Laravel must be praised for their default use of a modern password hashing algorithm, and directly integrated authentication management. Express.js and Spring require additional dependencies to do both, and especially Spring still provides support for MD5 and SHA-512 from the same interface as other safe password hashing algorithms.

When it comes to Authentication, Laravel excels at having pre-configured protection against brute-force attacks, something no other framework has. Both Laravel and Django have password requirements set up by default, with Django taking the better approach by having both length and complexity requirements. On the other hand, only Django includes a fully customizable password validator. No framework is logging failed login attempts by default, but Django and Laravel fire an event out of the box for failed logins. The session timeout for Django is by far the longest with 2 weeks, Laravel has 2 hours, Spring 30 minutes and Express.js should theoretically invalidate the session on browser exit. This would be the most reasonable default approach.

To protect against XSS attacks, all frameworks provide escaping of variables displayed through a template, Express.js requires libraries for it, and Spring needs a directive to be set. No framework provides context-based escaping of content. There exists a library for Laravel that does this, but this is something all frameworks lack.

Overall, Django and Laravel have the best protections configured and available out of the box. The other frameworks can sometimes make up for it with more dependencies and external libraries, but if security is a key aspect of the application - and it should be - those two should be preferred. That being said, for a number of reasons, additional factors play a role. First of all, there exist CVEs for all of the frameworks. [12]Additionally, through the installation of packages, further vulnerabilities can be introduced. Particularly NPM is known for security issues.[13] There also have been repeated attempts to steal encryption keys through packages that closely mimic the name of other commonly used package.[14].

[12]e.g. https://www.cvedetails.com/vulnerability-list/vendor_id-10199/product_id-18211/Djangoproject-Django.html

[13]See: https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities

[14]Source: https://www.zdnet.com/article/two-malicious-python-libraries-removed-from-pypi/

References

[1] The OWASP Foundation. OWASP Top 10 - 2017, The Ten Most Critical Web Application Security Risks. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf, 2017.

[2] Django Software Foundation. Django overview. https://www.djangoproject.com/start/overview/, 2019.

[3] Pivotal Software. Spring. https://spring.io/, 2019.

[4] Shahid Mansuri. Why Laravel is the Recommended Framework for Secure, Mission-Critical Applications. https://auth0.com/blog/why-laravel-is-the-recommended-framework-for-secure-mission-critical-applications/, 2019.

[5] IBM StrongLoop and other expressjs.com contributors. Express. https://expressjs.com/, 2019.

[6] Jean-Philippe Aumasson. Password Hashing Competition. https://password-hashing.net/, 2015.

[7] Django Software Foundation. Performing raw SQL queries: Executing custom SQL directly. https://docs.djangoproject.com/en/3.0/topics/db/sql/#executing-custom-sql-directly, 2019.

[8] Django Software Foundation. Password management in Django. https://docs.djangoproject.com/en/2.2/topics/auth/passwords/, 2019.

[9] Frank Rietta. What is the difference between bcrypt and SHA256? https://rietta.com/blog/bcrypt-not-sha-for-passwords/, 2016.

[10] Taylor Otwell. Authentication. https://laravel.com/docs/5.0/authentication, 2016.

[11] Leonid Shumakov. escape. https://github.com/laravelgems/escape, 2016.

[12] MITRE Corporation. Django Security Vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-10199/product_id-18211/Djangoproject-Django.html, 2019.

[13] npm. Auditing package dependencies for security vulnerabilities. https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities.

[14] zdnet. Two malicious Python libraries caught stealing SSH and GPG keys. https://www.zdnet.com/article/two-malicious-python-libraries-removed-from-pypi/, 2019.