Cheat Sheet

How to Develop Applications with Minimal Security Risks

Rudy De Busscher & Brian Vermeer

The Payara® Platform - Production-Ready, Cloud Native and Aggressively Compatible.

Contents

When we develop software we don't expect to be hacked or compromised. We build great new software for the needs of our clients. The people that use our software expect that our systems are safe and data will not be compromised. To ensure that safety we need to take responsibility and develop our applications in such a way that we can meet these expectations. Since the situation is real that our application is hacked or compromised.In this post, we discuss 7 pointers that can help you develop applications with a minimal security risk.

Know About Basic Code Exploits

A developer's job is to write code. First and foremost, the code should work and meet the functional requirements. However, most developers know that code should be maintainable, scalable and efficient. Finally, your code should be secure. A first step to do this is to get familiar with common security exploits. These exploits are common for a reason as they are typical mistakes that are made and still occur regularly. A few of these mistakes are:

SQL Injection

Since SQL Injection happens when you blindly use a value entered by the user into the query sent to the database, you need to avoid these kinds of operations at all costs. Instead, you need to validate and sanitize the received parameters. Using parameterized queries will also prevent this type of code exploits and should be the preferred way of performing queries.

When working in Java, we want to bind the parameters to, for instance, to a specific type. One of the ways to do this is by using a PreparedStatement like below.

```
statement.execute("DELETE FROM person WHERE id = " + personId);
```

Should be

```
PreparedStatement stmt =  connection.prepareStatement("DELETE FROM person WHERE
id = ?");
stmt.setInt(1, personId);
stmt.execute();
```

Another way to prevent this is to use frameworks like Hibernate, EclipseLink, or Spring data that do the heavy lifting for you. However, whenever you are writing custom queries like HQL in Hibernate, yet again this problem can occur. So be careful.

XXS (Cross Site Scripting)

The Cross-site Scripting vulnerability is similar to the SQL injection one but is related to the web screens. When data entered by the user is shown on the web pages later on, like comments in a forum, they can contain additional script values that interact with the browser and user when shown on the pages.

To avoid it, data entered by the user needs to be validated and sanitized so that you do not store some content with malicious intent.

An option is to use the Jakarta Bean Validation project. It has many annotations for validating the input when using it combined with Jakarta EE compatible product like Payara Server, or you can do a programmatic validation with Hibernate Validator using the ESAPI framework for example.

```
@Pattern(regexp = "^(?!.*<.*$)")
private String name;
```

Will allow only values that do not contain *'<'* as it is unusual in a name and related to *<script>* entries associated with XSS.

DOS (Denial of Service)

Each request received by the backend requires resources from that server. When it receives many more requests than the system is designed for, it will delay the response for genuine requests, or in the worst scenario, it will halt your service.

So at a minimum, make sure that you have a limitation on the number of requests you process concurrently. The Thread Pool in Java used in many servers is a good example of limiting the processing and avoiding that your service stops due to resource exceptions.

More advanced functionality is required to limit the number of requests from each client so that the resources are divided more fairly amongst the users of your application.

A possible countermeasure against a DOS could be rate-limiting. You limit the number of requests to your backend during a period, say ten requests per second. This can be based on IP address, account, or whatever fits your needs. If the number of requests exceeds the limit, you will return an error like an HTTP 429 (Too Many Requests) response.

Scan your Application for Known Vulnerabilities

Most applications contain not only code written by your application developers. In many cases, your applications heavily depend on third-party frameworks and libraries. These frameworks and libraries can have Security vulnerabilities. As your team is responsible for the complete application, including the imported libraries and frameworks, you should be aware of possible security issues. Therefore you should check if there are any vulnerabilities reported for the version you are using. A great option is to use Snyk for scanning for instance, on your local machine, and in your CI pipeline. It will provide you information about vulnerabilities you inherit from your dependencies and how you can remediate them.

When to Check

You should check as often as possible to verify if there are no new security Vulnerabilities reported for the version you use. You can integrate that in your build process, so that every time a build runs, the check is also performed. Also, make sure that you monitor your dependencies once your application is in production. Security vulnerabilities are discovered over time, and you want to be alerted sooner rather than later. Snyk can also monitor this for you.

Create a Dependency Management Strategy

Using third-party libraries and frameworks to accelerate your development process is a common practice. However, once a package is part of your system it rarely gets updated or removed. Therefore it is important to create a dependency management strategy containing the following three aspects. Pick, Update and Remove

How to Pick your Dependencies

When including a package in your system, it is important to think of several things. As there are many open source libraries available, the quality may vary heavily. Consider:

- Does it solve the whole problem or just a part
- Is it worth importing the complete library, or am I just using a very small part of it?
- How is the dependency health? Think of maintainers, release cadence number of open issues.
- Do I need this dependency, or is the functionality already available via a library I already included.
- Does the library contain security vulnerabilities?

Updating your Dependencies

In general, it is a best practice to upgrade the frameworks and libraries for each minor version update that becomes available. Those should be backward compatible and provide you with general bug and security fixes.

The same rules apply as for picking a dependency. We need to check this regularly and act accordingly. Updating early and often will most likely prevent this. Your package manager like Maven and Gradle can help you find out if there are newer versions available.

You shouldn't forget your runtime either as that will also receive regular updates. These contain new features, bug fixes but also security vulnerability fixes. The Payara Platform products for example, are released monthly so that users can keep their environment secure.

Removing Dependencies

If you don't use a library anymore, you should remove it. This sounds straightforward but rarely happens. Although the code from an unused dependency will not be actively called, it is still available in the classpath. This means that when a security vulnerability is present, it can still be harmful depending on the issue. So clean up these dependencies and stay safe.

Validate your Configuration

What Endpoints are Available?

As a developer, we have a pretty good overview of all the endpoints we expose for the application. But we are not always aware of the endpoints made available by the frameworks or the runtimes we use.

These endpoints can often contain some sensitive information about your system so that it facilitates the job of the hacker.

So make sure that you have always a good inventory of all the endpoints that are exposed, who is allowed to access them, and that you take protective measures, like shielding them through the load balancer or apply adequate authentication to them.

Hijacking Insecure Sonnections

When your application uses an unsecured, this is not using the SSL protocol and https, connection, information can be read very easily and allows redirects to fake sites.

Any data sent over an HTTP connection can easily be viewed when transferred to the server. This means that you should always make use of a secure connection so that all information is encrypted. But using a secure connection is also important for avoiding a man-in-the-middle attack. A simple permanent redirect returned by a hacker can make all the communication go through a hacker-controlled domain. The application seems to work normally but all the information you type in is intercepted.

Security Headers

Besides all the measures you take regarding secure connections and authentication of your users, 5 headers should be defined by all applications.

The HTTP Strict Transport Security, Content Security Policy, Cross-Site Scripting Protection, X-Frame-Options, and X-Content-Type-Options make sure that your application and communication has the highest security measure implemented.

It helps you deal with Cross-Site scripting, Mime sniffing, and clickjack attacks.

A great tool you can use to check the security headers for your webpage is securityheaders.com

Develop a Code Review Strategy

Reviewing code is a common practice in many development teams and a good practice for several reasons. But the review should not be limited to aspects like best practices, application functionality, and code readability but should also include security aspects.

Create a checklist so that you are reminded of the things you should look at during a review. For the security part, things like hardcoded passwords, insecure parameter handling, etc ... should be on the list in any case.

A great way to start is checking out the code review cheat sheet and see what you can apply today.

Who Reviews the Code?

Be aware not to rush your code review process. This means that when you create a new piece of code you should schedule a reasonable amount of time for the reviewer. A proper review is important to not only find bugs, but also prevent security issues before deploying the software to production.

Next to this, be aware who you are asking for a review. If a piece of code is havely security focused maybe you can ask the person who is most qualified to do the review. This way you will get more value out of the review. In larger teams, it might be a good idea that a code review is performed by several people. One can be focussing more on the functional aspects of the code and other people can concentrate on the security part. Picking the right people for the right review is key. When doing code reviews right it can be a great addition to your secure development mindset.

In addition, when there are no dedicated pen-testers or security champions available, the reviewers can even perform the pen testing and other security-related testing. In an ideal world this should be done by someone with specific expertise. However, this can be a more pragmatic way to make sure your applications withstand the attacks once it is released.

Identify Vulnerable Information

What Information is Vulnerable?

In most applications, not all data has the same vulnerability. Username and password and personal data, especially medical information are of course very sensitive information. Other information, like the list of all city names, is not interesting at all for hackers.

So you must have a good inventory of which data is sensitive and vulnerable and what other data is less important. This can help you to focus on those areas of your applications containing sensitive data. Even if a piece of data does not seem to be important on its own, combining it with other information can result in privacy issues. For instance, a street name on its own might not look valuable unless you connect it to the person who lives there.

Of course, this should not lead to situations where the basic security rules aren't followed so that your front door is maybe highly secured but the side door is wide open and bypassing all security measures.

What Data do we Need?

Determining what information is required for a certain user role or screens is not only beneficial from a technical point of view but also security-wise. You do not want to display internal id's in your endpoints for instance as these can help attackers to connect pieces of data.

Sending too much data to the client application requires more processing time and bandwidth, but it might also contain data that is not required at that time. This additional information can be sensitive and thus we have a greater risk of exposing this information to people that should not see it.

Based on what information is regarded as sensitive and is determined as required, you can make sure that your message contains just enough information and thus limits that sensitive data is misused.

It is wise to create a separation between the pure data entities and the data that you display. Although it might be convenient to display a database entry directly, using an extra layer with for instance a view object prevents accidental exposure.

Follow Clean Code Rules

Clean code is something we all strive for. Following standard code conventions regardless of the language, you are using will make your code better readable. Better readable code means easier to maintain. You might think that this has nothing to do with security but the opposite is true.

If code is easier to read and understand, it is also easier to sport bugs and security issues. This means that reviews will be easier and if a security issue occurs, it will be more straightforward to solve it. Furthermore, most of the clean code rules already include security best practices. If the code quality is considered high, the possibility of an accidental security issue is far less.

Static Analysis

To help a developer with following the clean code guidelines we can use tools like a linter. Tools like PMD, checkstyle and spotbugs are a great addition to punt you to code smells and possible security bugs. But there are tons of linters and static analysis tools for a variety of languages you can use. Make sure that the developers in your team are using a common code convention standard. Also, use static analysis tools/ linters to automatically check your code before publishing it.

Summary

The security aspects of your application are the most important non-functional requirement. Every day, more than 20,000 websites get hacked and on average, 75 data records are stolen each second from applications.

Knowing the basic security vulnerabilities, like SQL injection and XSS attacks is a must. As a minimum, you should be familiar with the top 10 security vulnerabilities as reported by OWASP.

But it is not only the code that you write that can be the problem. Your application uses 3rd party dependencies and a runtime. They can also contain vulnerabilities. So, make sure you have a proper review of those dependencies, and a process to upgrade them regularly to address vulnerabilities that get fixed.

Each developer should be aware of the security aspects of the code he or she writes. But you need a dedicated team of people working in this area. People who do the security testing, including pen testing, reviewing code from a security point of view, identify sensitive data, documentation all endpoints and their security requirements.

sales@payara.fish **+44 207 754 0481** **www.payara.fish**