# Chapter 20. Securing Modern Web Applications

Up to this point, we have spent a significant amount of time analyzing techniques that can be used for researching, analyzing, and breaking into web applications. These preliminary techniques are important in their own right and give us important insights as we move into the third and final part of this book: defense.

Today's web applications are much more complex and distributed than their predecessors. This opens up the surface area for attack when compared to older, monolithic web applications—in particular, those with server-side rendering and little to no user interaction. These are the reasons I structured this book to start with recon, followed by offense, and finally defense.

I believe it is important to understand the surface area of a web application and understand how such a surface area can be mapped and analyzed by a potential hacker. Beyond this, I believe that having an understanding of techniques hackers are using to break into web applications is also crucial knowledge for anyone looking into securing a web application. By understanding the methodology a hacker would use to break into your web application, you should be able to derive the best ways to prioritize your defenses and camouflage your application architecture and logic from malicious eyes.

All of the skills and techniques we have covered up until this point are synergistic. Improving your mastery of recon, offense, or defense will result in extremely efficient use of your time.

Defending a web application is somewhat akin to defending a medieval castle. A castle consists of a number of buildings and walls, which represent the core application code. Outside of the castle are a number of buildings that integrate with and support the castle's owner (usually a lord) in a way that describes an application's dependencies and integrations. Due to the large surface area in a castle and the surrounding kingdom, in wartime it is essential for defenses to be prioritized; it would be infeasible to maximize the defensive fortifications at every potential entrance point.

In the world of web application security, such prioritization and vulnerability management is often the job of security engineers in large corporations or more generalized software engineers in smaller companies. These professionals take on the role of master defender, using software engineering skills in combination with recon and hacking skills to reduce the probability of a successful attack, mitigate potential damages, and then manage active or past damages.

## Defensive Software Architecture

The first step in writing a well-fortified web application starts prior to any software actually being written. This is the architecture phase. In the architecture phase of any new product or feature, deep attention to detail should be spent on the data that flows throughout the application.

It could be argued that most of software engineering is efficiently moving data from point A to point B. Similarly, most of security engineering is efficiently securing data in transit from point A to point B and wherever it may rest before, after, or during that transit.

It is much easier to catch and resolve deep architectural security flaws before actually writing and deploying the software. After an application has been adopted by users, the depth at which a re-architecture can be performed to resolve a security gap is often limited.

This is especially true in any type of web application that consumers build upon. Web applications that allow users to open their own stores, run their own code, and so on can be extremely costly to re-architect because deep re-architecture may require customers to redo many time-significant manual processes.

In the following chapters, we will learn a number of techniques to properly evaluate the security in an application's architecture. These techniques range from analysis of data flow to threat modeling for new features.

## Comprehensive Code Reviews

During the process of actually writing a web application that has already been evaluated as a secure architecture, the next step is carefully evaluating each commit prior to release into the codebase. Most companies have already adopted mandatory code review processes to improve quality assurance, reduce technical debt, and eliminate easy-to-find programming mistakes.

Code reviews are also a crucial step in ensuring that released code meets security standards. To reduce conflict of interest, commits to source code version control should not only be reviewed by members of the committer's team but also by an unrelated team (especially in regard to security).

Catching security holes at the code review level on a per-commit basis is actually easier than one would think. The major points to look out for are:

- How is data being transmitted from point A to point B (typically over a network, and in a specific format)?
- How is data being stored?
- When data gets to the client, how is it presented to the user?
- When data gets to the server, what operations occur on it and how is it persisted?

In the following chapters, we will evaluate significantly more specific measures for performing security code reviews. But this list provides a basis from which to build upon and prove that anyone can get started reviewing for security.

## Vulnerability Discovery

Assuming your organization and/or codebase has already undertaken steps to evaluate security before writing code (architecture) and during the development process (code reviews), the next step is finding vulnerabilities in the code that occur as a result of bugs that are not easily identifiable (or missed) in the code review process. Vulnerabilities are found in a number of ways, and some of these ways will damage your business/reputation, while others will not.

The old-fashioned way of finding vulnerabilities is either by customer notification or (worst case) widespread public disclosure. Unfortunately, some companies still rely on this as their only means of finding vulnerabilities to fix in their web applications.

More modern methods for finding vulnerabilities exist, and they can save your product from a wave of bad PR, lawsuits, and loss of customers. Today's most security-conscious companies use a combination of the following:

- Bug bounty programs
- Internal red/blue teams
- Third-party penetration testers
- Corporate incentives for engineers to log known vulnerabilities

By making use of one or more of these techniques to find vulnerabilities before your customers or the public do, a large corporation can save huge amounts of money with a little bit of expense up front.

We will evaluate each of these methods of finding vulnerabilities in the following chapters. We will also analyze several well-known cases of companies that did not properly invest in such proactive security measures, and the huge financial losses that stemmed from such negligence.

## Vulnerability Analysis

After finding a vulnerability in your web application, there are several steps that should be taken to properly triage, prioritize, and manage that vulnerability.

First off, not all vulnerabilities carry as much risk as others. It is a well-known fact in security engineering that some vulnerabilities are worth pushing off until developers have free time, whereas others are worth dropping all current development processes in order to patch.

The first step in vulnerability management is evaluating the risk a vulnerability presents to your company. The risk level of a vulnerability determines the priority required when determining when and in what order to fix vulnerabilities.

Risk and priority can be derived from:

- Financial risk to the company
- Difficulty of exploitation
- Type of data compromised
- Existing contractual agreements
- Mitigation measures already in place

After determining the risk and prioritization of a vulnerability, the next step involves developing tracking methods to ensure the solution is progressing in a timely manner and alongside your contractual obligations. The final step is writing automated tests to ensure the vulnerability does not regress and reopen after a fix is deployed.

## Vulnerability Management

After assessing the risk of a vulnerability, and prioritizing it based on the factors listed, a fix must be tracked through to completion. Such fixes should be completed in a timely manner, with deadlines determined based off of the risk assessment. Furthermore, customer contracts should be analyzed in response to an assessed vulnerability to determine if any agreements have been violated.

Also, if the vulnerability can be recorded during this time frame, additional logging should be put in place to ensure that no hacker attempts to take advantage of the vulnerability while the fix is being developed. Lack of logging for known vulnerabilities has led to the demise of several companies that were not aware a vulnerability was being abused while they waited for resolution from their engineering teams.

Managing vulnerabilities is an ongoing process. Your vulnerability management process should be carefully planned out and written down so that your progress can be recorded. This should result in more accurate timelines as time goes on and time-to-fix burn rates can be averaged.

## Regression Testing

After deploying a fix that resolves a vulnerability, the next step is to write a regression test that will assert that the fix is valid and the vulnerability no longer exists. This is a best practice that is not being used by as many companies as it should be. A large percentage of vulnerabilities are regressions—either directly reopened bugs or variations of an original bug. A security engineer from a large software company (10,000+ employees) once told me that approximately 25% of their security vulnerabilities were a result of previously closed bugs regressing to open.

Building and implementing a vulnerability regression management framework is simple. Adding test cases to that framework should take a small fraction of the time that an actual fix took. Vulnerability regression tests cost very little up front but can save huge amounts of time and money in the long run. We will be discussing how to effectively build, deploy, and maintain a regression testing framework in the following chapters.

## Mitigation Strategies

Finally, an overall best practice for any security-friendly company is to actively make a good effort to mitigate the risk of a vulnerability occurring in the application codebase. This is a practice that happens all the way from the architecture phase to the regression testing phase.

Mitigation strategies should be widespread, like a net trying to catch as many fish as possible. In crucial areas of an application, mitigation should also run deep.

Mitigation comes in the form of secure coding best practices, secure application architecture, regression testing frameworks, secure software development life cycle (SSDL), and secure-by-default developer mindset and development frameworks. Throughout the following chapters, we will learn a number of ways to mitigate and sometimes eliminate the risk that a particular vulnerability can introduce into our codebase.

Practicing all of the preceding steps will greatly enhance the security of any codebase you work on. It will eliminate huge amounts of risk from your organization and save large amounts of money while protecting you from the huge brand damage that would occur otherwise in due time.

## Applied Recon and Offense Techniques

The techniques we learned in Parts I and II are not required prior to progressing into Part III. However, deep knowledge of recon and offensive techniques will give you insight into building stronger defenses that could not be obtained otherwise.

As we progress through the process of securing a web application, keep in mind the recon techniques learned from Part I. These techniques will give you insight into how to camouflage your application from unwanted eyes. They will also give you insight as to how to prioritize fixes because you will note that some vulnerabilities will be easier to find than others.

The material from Part II will also be valuable throughout this section. By understanding common vulnerabilities that hackers look for in order to break into a web application, you will better understand what types of defenses you can put up to mitigate such attacks. Knowledge of specific categories of exploit should also help you prioritize your fixes because you will understand what type of data will be put at risk if one of these exploits is found in your web application.

This book is not a comprehensive know-all reference, but it should provide enough foundational knowledge for you to seek more information on recon, offense, and defense. It should give you the foundation you need to understand how to communicate about recon techniques, vulnerabilities, and mitigation methods. With this knowledge in hand, you should be able to easily accelerate your learning in the realm of software security and begin directing your own self-studies into whichever security realm you want to master.

## Summary

Securing modern web applications requires a wide range of skills. First, web applications must be designed in a way that makes them architecturally secure. Next, as code is written, it must undergo processes such as the security code review prior to being released into a production build. Next, systems for discovering, analyzing, and managing vulnerabilities need to be introduced. Finally, regression test frameworks and workflows need to be updated to ensure the same security gap never occurs more than once.