Preface

Welcome to *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*. In this preface, we will discuss the book's content and who this book is for, including the skill sets required to make the most of the technical content in the following chapters. Reading the preface will help you understand if this book is for you.

Changes from the First Edition

You will find a significant number of changes when comparing this book to its prior first edition. There are over one hundred pages of new content, but beyond that there are dozens of edited pages.

The first edition was primarily focused at the entry- and mid-level engineer, but feedback often requested more advanced content from which you could continue down a particular learning path for each chapter. Most chapters now have advanced content offered, and as such my hope is that senior security professionals will now benefit more from reading this book.

Additionally, the book has had a significant amount of updates to incorporate recent technologies. I felt it imperative to add example cases and code for securing and attacking new but common forms of technology in web applications, for example GraphQL and NoSQL databases.

The second edition has significant swaths of new security content including content covering the latest and most popular web application technologies. It also has been modified to include more advanced content per chapter and to incorporate dozens, if not hundreds, of reader and editor suggestions and requests within its pages.

I hope you find this book well organized and enjoyable to read, and that once you have finished it, you walk away with new knowledge and perspectives that enhance your information security skill set.

Prerequisite Knowledge and Learning Goals

This is a book that will not only aid you in learning how to defend your web application against hackers, but will also walk you through the steps hackers take in order to investigate and break into a web application.

Throughout this book we will discuss many techniques that hackers are using today to break into web applications hosted by corporations, governments, and occasionally even hobbyists. Following sufficient investigation into the previously mentioned techniques, we begin a discussion on how to secure web applications against these hackers.

In doing so you will discover brand-new ways of thinking about application architecture. You will also learn how to integrate security best practices into an engineering organization. Finally, we will evaluate a number of techniques for defending against the most common and dangerous types of attacks that occur against web applications today.

After completing *Web Application Security*, you will have the required knowledge to perform recon techniques against applications you do not have code-level access to. You will also be able to identify threat vectors and vulnerabilities in web applications and craft payloads designed to compromise application data, interrupt execution flow, or interfere with the intended function of a web application.

With these skills in hand, and the knowledge gained from the final section on securing web applications, you will be able to identify risky areas of a web application's codebase and understand how to write code to defend against attacks that would otherwise leave your application and its users at risk.

NOTE

The content in this book ramps up progressively, so if you choose to skip ahead and find you are missing essential prerequisite information, just go back a few chapters to catch up. Any topics that are not defined as a prerequisite in this preface should not be presented in the book without prior explanation.

Why Are Examples in JavaScript?

The majority of code samples in this book are written in JavaScript, which is atypical for software security–related books. You may wonder why I chose this when planning the book.

All good security engineers *must understand JavaScript* because it is the only programming language that can be run on the client inside of a web browser at this point in time. Therefore, I decided it would be better to present most of the server-side examples in JavaScript, given the client-side examples have to be in JavaScript. Luckily, programming languages are mostly interchangeable, so if you are capable of understanding the content conceptually you should be able to apply any of the server-side attacks or mitigations against any other server-side programming language.

Why Teach Concepts Instead of Tools?

This book explains offensive and defensive cybersecurity techniques at a reasonably low level to reveal what happens under the hood of popular tools. For example, in this book you will develop Cross-Site Scripting (XSS) payloads and find XSS sinks rather than utilize an automated XSS discovery tool like XSS-Sniper.

Many security engineers and pen testers today do not understand what occurs underneath the tools they use on a regular basis. The downside to this is that reliance on tools results in said engineers and pen testers not being able to take advantage of advanced attacks or mitigations. Furthermore, tools in the application security industry in particular are

ephemeral—that is, they are swapped out and become obsolete on a regular basis.

By teaching the underlying concepts rather than relying on paid tooling, anyone who reads this book should be able to hop from one tool to another or deploy custom tools, attacks, or mitigations. This would not be possible if the book focused on specific tooling.

Suggested Background

The book's audience is quite broad. It is written and structured for anyone with an intermediate-level background in software engineering. What is an "intermediate-level background in software engineering"? The answer will differ significantly from person to person.

For a highly technical person, this book might actually require only a beginner-level background in software engineering. In other words, a system administrator with prior web development and/or scripting experience (if sufficient) could reasonably read through this book and understand all of the examples. That said, this book includes examples that require both client and server coding knowledge. Knowing one or the other is not sufficient for a deep understanding of these examples.

This book also includes discussions regarding basic client/server networking over HTTP. Additionally, conversations regarding software architecture pop up in later chapters as we explore ways of integrating in-house software with third-party software while mitigating security risks.

Because so many topics are covered in this book, I have chosen to define the required skill level to successfully complete this book as "intermediate" versus "beginner." This book would not be appropriate for those without any experience or knowledge of writing production-quality software applications.

Minimum Required Skills

In this book, an "intermediary-level background in software engineering" implies the following:

- You can write basic CRUD (create, read, update, delete) programs in at least one programming language.
- You can write code that runs on a server somewhere (such as back-end code).
- You can write at least some code that runs in a browser (frontend code, usually JavaScript).
- You know what HTTP is, and can make, or at least read, GET/POST calls over HTTP in some language or framework.
- You can write, or at least read and understand, applications that make use of both server-side and client-side code, and communicate between the two over HTTP.
- You are familiar with at least one popular database (MySQL, MongoDB, etc.).

These skills represent the minimum criteria for successfully following the examples in this book. Any experience you have beyond these bullet points is a plus and will make this book that much easier for you to consume and derive educational value from.

NOTE

Although the majority of the code examples in this book are written in JavaScript for simplicity's sake (so that the client and server code are in the same language), most of the examples can be applied to other languages with little effort.

I have done my best to organize the topics in this book so that they ramp up in difficulty at a maintainable pace. I have also tried to be as verbose as possible in my explanations. This means that whenever I cover a new technology, I start with a brief background and overview of how that technology works.

Who Benefits Most from Reading This Book?

Prerequisite skills aside, I believe it is important to clarify who will benefit from this book the most, so I'd like to explain who my target audience is. To do so I have structured this section in terms of learning goals and professional interests. If you don't fit into one of the following categories, you can still learn many valuable or at least interesting concepts from this book.

This book was written to stand the test of time, so if you decide later on to pursue one of the occupations in its target audience, all of the knowledge from this book should still be relevant.

Software Engineers and Web Application Developers

The primary audience for this book is an early- to mid-career software engineer or web application developer. Ideally, you are interested in gaining a deep understanding of either offensive techniques used by hackers or defensive techniques used by security engineers to defend against hackers.

Often the titles "software engineer" and "web application developer" are interchangeable, which might lead to a bit of confusion. I use both of them throughout the book. Let's start off with some clarification.

Software engineers

When I use the term *software engineer*, I am referring to a generalist who is capable of writing software that runs on a variety of platforms.

Software engineers will benefit from this book in several ways. First off, much of the knowledge in this book is easily transferable to software that does not run on the web. It is also transferable to other types of net-

worked applications, with native mobile applications being the first that come to mind.

Several exploits discussed in this book take advantage of server-side integrations involving communication with a web application and another software component. As a result, it is safe to consider any software that interfaces with a web application as a potential threat vector (databases, CRM, accounting, logging tools, etc.).

Web application developers

In this book, a *web application developer* is someone who is highly specialized in writing software that runs on the web. They are often further subdivided into frontend, backend, and full stack developers.

Historically, many attacks against web applications have targeted server-side vulnerabilities. As a result, I believe this book's use case for a backend or full stack developer is clear. I also believe this book to be valuable for other types of web application developers, including those who do not write code that runs on a server, but instead runs on a web browser (frontend/JavaScript developers).

As I will explain, many of the ways hackers take advantage of today's web applications originate via malicious code running in the browser. Some hackers are even taking advantage of the browser DOM or CSS stylesheets in order to attack an application's users. It is important for frontend developers who do not write server-side code to be aware of the security risks their code may expose and how to mitigate those risks.

General Learning Goals

This book should be a fantastic resource for anyone matching the aforementioned descriptions who is looking to make a career change to a more security-oriented role. It will also be valuable for those looking to learn how to beef up the defenses in their own code or in the code maintained by their organization.

If you want to defend your application against very specific exploits, this book is also for you. This book's structure should enable you to use it as a security reference without ever having to read any of the chapters that involve hacking. That is, of course, if that is your only goal. I suggest reading from cover to cover for the best learning experience, but if you are looking only for a reference on securing against specific types of hacks, flip halfway through and get started.

Security Engineers, Pen Testers, and Bug Bounty Hunters

The book's structure also lends itself to being used as a resource for penetration testing, bug bounty hunting, or any other type of application-level security work. If that is relevant or interesting to you, you may find the first half of the book more to your liking.

We will take a deep dive into how exploits work from both a code level and an architectural level rather than simply executing well-known open source software (OSS) scripts or making use of paid security automation software. Because of this, there is a second audience for this book—software security engineers, IT security engineers, network security engineers, penetration testers, and bug bounty hunters. This book will be beneficial to security professionals who understand how many attacks work conceptually but would like to delve into the systems and code behind a tool or script.

TIP

Want to make a little bit of extra money on the side while developing your hacking skills? Read this book and then sign up for one of the bug bounty programs noted in Part III. This is a great way to help companies improve the security of their products while developing your hacking skills and making some additional cash.

Today it is commonplace for penetration testers to operate using a wide array of prebuilt exploit scripts. This has led to the creation of many paid

and open source tools that automate classic attacks, and attacks that can be easily run without deep knowledge regarding the architecture of an application or the logic within a particular block of code.

The exploits and countermeasures contained within this book are presented without the use of any specialized tools. Instead, we will rely on our own scripts, network requests, and the tooling that comes standard in Unix-based operating systems, as well as the standard tooling present in the three major web browsers (Chrome, Firefox, and Edge). This is not a judgment on the value of specialized security tools—I think that many of them are exceptional and make delivering professional, high-quality penetration tests much easier!

This book does not make use of specialized security tools so that we can focus on the most important parts of finding a vulnerability, developing an exploit, prioritizing data to compromise, and making sure you can defend against all of the above. As a result, I believe that by the end of this book you will be prepared to go out into the wild and find new vulnerabilities, develop exploits against systems that have never been exploited before, and harden the most complex systems against the most persistent attackers.

How Is This Book Organized?

You will soon find that this book is structured quite differently than most other technology books out there. This is intentional. This book is purposefully structured so that there is a nearly 1:1 ratio of chapters regarding hacking (offense) and security (defense).

We begin our adventure with a bit of a history lesson and some exploration into the technology, tools, and exploits of the past. Then we move on to our main topic: exploitation and countermeasures for modern web applications. (Hence the subtitle of this book.)

The main content is divided into three major parts, each containing many individual chapters covering a wide array of topics. Ideally, you would venture through this book in a linear fashion, from page one all the way

to the final page. As mentioned, reading this book in order will provide the greatest learning possible, but it can also be used as a hacking reference or a security engineering reference by focusing on the first or second half, respectively.

By now you should understand how to navigate the book, so let's go over the three main parts to grasp the importance of each.

Recon

The first part of this book is "Recon," where we evaluate ways to gain information regarding a web application without necessarily trying to hack it. In "Recon," we discuss a number of important technologies and concepts that are essential to master if you wish to become a hacker. These topics will also be important to anyone looking to lock down an existing application because the information exposed by many of these techniques can be mitigated with appropriate planning.

I have had the opportunity to work with what I believe to be some of the best penetration testers and bug bounty hunters in the world. Through my conversations with them and my analysis of how they do their work, I've come to realize this topic is much more important than many other books make it out to be. For many of the top bug bounty hunters in the world, expert-level reconnaissance ability is what differentiates "great" hackers from "good" hackers. In other words, it's one thing to have a fast car (in this case, perhaps knowing how to build exploits), but without knowing the most efficient route to the finish line, you may not win the race. A slower car could make it to the finish line in less time than a fast one if a more efficient path is taken.

If fantasy-based analogies hit closer to home, you could think of recon skills as something akin to a rogue in a role-playing game. In our case, the rogue's job isn't to do lots of damage, but instead to scout ahead of the group and circle back with intel. It's the character who helps line up the shots and figures out which battles will have the greatest rewards.

The last part in particular is exceedingly valuable because it's likely many types of attacks could be logged against well-defended targets. This means

you might only get one chance to exploit a certain software hole before it is found and closed. We can safely conclude that the second use of reconnaissance is figuring out how to prioritize your exploits.

If you're interested in a career as a penetration tester or a bug bounty hunter, Part I will be of utmost importance to you. This is largely because tests are performed "black box" style in the world of bug bounty hunting, and to a lesser extent penetration testing. "Black box" testing is a style of testing where the tester has no knowledge of the structure and code within an app and, hence, must build their own understanding of the application through careful analysis and investigation.

Offense

Part II is "Offense." Here the focus of the book moves from recon and data gathering to analyzing code and network requests. Then with this knowledge we will attempt to take advantage of insecurely written or improperly configured web applications.

WARNING

A number of the chapters explain actual hacking techniques used by malicious black hat hackers in the real world.[1] It is imperative that if you are testing techniques found in this book, you do so only against an application that you own or have explicit written permission to test exploits against.

Improper usage of the hacking techniques presented in this book could result in fines, jail time, etc., depending on your country's laws on hacking activity.

In Part II, we learn how to both build and deploy exploits. These exploits are designed to steal data or forcibly change the behavior of an application. This part of the book builds on the knowledge from Part I, "Recon." Using our previously acquired reconnaissance skills in conjunction with newly acquired hacking skills, we will begin taking over and attacking demo web applications.

Part II is organized on an exploit-by-exploit basis. Each chapter explains in detail a different type of exploit. These chapters start with an explanation of the exploit itself so you can understand how it works mechanically. Then we discuss how to search for vulnerabilities where this exploit can be applied. Finally, we craft a payload specific to the demo application we are exploiting. We then deploy the payload and observe the results.

XSS, one of the first exploits we dig into, is a type of attack that works against a wide array of web applications, but it can be applied to other applications as well (e.g., mobile apps, Flash/ActionScript games, etc.). This particular attack involves writing some malicious code on your own machine, then taking advantage of poor filtration mechanisms in an app that will allow your script to execute on another user's machine.

When we discuss an exploit like an XSS attack, we will start with a vulnerable app. This demo app will be straightforward and to the point, ideally just a few paragraphs of code. From this foundation, we will write a block of code to be injected as a payload into the demo app, which will then take advantage of a hypothetical user on the other side.

Sounds simple, doesn't it? And it should be. Without any defenses, most software systems are easy to break into. As a result, with an exploit like XSS where there are many defenses, we will progressively dig deeper and deeper into the specifics of writing and deploying an attack.

We will initially attempt to break down routine defenses and eventually move on to bypassing more advanced defense mechanisms. Remember, just because someone built a wall to defend their codebase doesn't mean you can't go over it or underneath it. This is where we will get to use some creativity and find some unique and interesting solutions.

Part II is important because understanding the mindset of a hacker is often vital for architecting secure codebases. It is exceptionally important for anyone interested in hacking, penetration testing, or bug bounty hunting.

Defense

The third and final part of this book, "Defense," is about securing your own code against hackers. In Part III, we go back and look at every type of exploit we covered in Part II and attempt to consider them again with a completely opposite viewpoint. Here, we will not concentrate on breaking into software systems; we will attempt to prevent or mitigate the probability that a hacker could break into our systems.

In Part III you will learn how to protect against specific exploits from Part II, in addition to learning general protections that will secure your codebase against a wide variety of attacks. These general protections range from "secure by default" engineering methodologies to secure coding best practices that can be enforced easily by an engineering team using tests and other simple automated tooling (such as a linter). Beyond learning how to write more secure code, you will also learn a number of increasingly valuable tricks for catching hackers in the act and improving your organization's attitude toward software security.

Most chapters in Part III are structured somewhat akin to the hacking chapters in Part II. We begin with an overview of the technology and skills required as we begin preparing a defense against a specific type of attack.

Initially we will prepare a basic-level defense, which should help mitigate attacks but may not always fend off the most persistent hackers. Finally, we will improve our defenses to the point where most, if not all, hacking attempts will be stopped.

At this point, the structure of Part III begins to differ from that of Part II as we discuss trade-offs that result from improving application security. Generally speaking, all measures of improving security will have some type of trade-off outside of security. It may not be your place to make suggestions on what level of risk should be accepted at the cost of your product, but you should be aware of the trade-offs being made.

Often, these trade-offs come in the form of application performance. The more efforts you take to read and sanitize data, the more operations are

performed outside of the standard functionality of your application. Hence a secure feature typically requires more computing resources than an insecure feature.

With further operations also comes more code, which means more maintenance, tests, and engineering time. This development overhead to security often comes in the form of logging or monitoring overhead as well. Finally, some security precautions will come at the cost of reduced usability.

A very simple example of this process of comparing security benefits to their cost, in terms of usability and performance, is a login form. If an error message for an invalid username is displayed to the user when attempting to log in, it becomes significantly easier for a hacker to brute force username:password combinations. This occurs because the hacker no longer has to find a list of active login usernames, as the application will confirm a user account. The hacker simply needs to successfully brute force a few usernames, which can be confirmed and logged for later break-in attempts.

Next, the hacker only needs to brute force passwords rather than username:password combinations, which implies significantly decreased mathematical complexity and takes much less time and resources.

Furthermore, if the application uses an email and password scheme for login rather than a username and password scheme, then we have another problem. A hacker can use this login form to find valid email addresses that can be sold for marketing or spam purposes. Even if precautions are taken to prevent brute forcing, carefully crafted inputs (e.g., *first.last@company.com*, *firstlast@company.com*, *firstl@company.com*) can allow the hacker to reverse engineer the schema used for company email accounts and pinpoint the valid accounts of execs for sales or individuals with important access criteria for phishing.

As a result, it is often considered best practice to provide more generic error messages to the user. Of course, this change conflicts with the user experience because more specific error messages are definitely ideal for the usability of your application.

This is a great example of a trade-off that can be made for improved application security, but at the cost of reduced usability. This should give you an idea of the type of trade-offs that are discussed in Part III of this book.

This part of the book is extremely important for any security engineer who wants to beef up their skills, or any software engineer looking at transitioning to a security engineering role. The information presented here will help in architecting and writing more secure applications.

As in Part II, understanding how an application's security can be improved is a valuable asset for any type of hacker. This is because while routine defenses can often be easily bypassed, more complex defenses require deeper understanding and knowledge to bypass. This is further evidence as to why I suggest reading the book from start to finish.

Although some parts of this book may give you more valuable education than others, depending on your goals, I doubt any of it will be wasted. Cross-training of this sort is particularly valuable, as each part of the book is just another perspective on the same puzzle.

Language and Terminology

It has probably become evident by now that this book aims to teach you a number of very useful but also very rare and particular skills. While these skills are increasingly valuable, and will much improve your salability on the job market, they are also quite difficult to learn, requiring focus, aptitude, and the capacity to pick up a whole new mental model that defines how you look at web applications.

To successfully communicate these new skills, we need to establish some common language. This is important to avoid confusion and to help you express your new ideas in a way that is consistent across security and engineering organizations.

Each time I introduce a new term or phrase, I do my best to explain it. In particular, when dealing with acronyms, I spell out the acronym first

prior to using the acronym by itself. You saw this earlier when I spelled out XSS. Beyond that, I have done my best to determine what terms and phrases might need explaining. I have collected them and organized them into the following tables (Tables P-1 to P-3). If you stumble across a term or phrase you don't fully understand, jump back to this preface (bookmark it!) to see if it is listed here. If it isn't, feel free to email my editor, and perhaps we can include it in the next edition of the book.

Table P-1. Occupation

Occupation	Description
Hacker	Someone who breaks into systems, typically in order to exfiltrate data or cause the system to perform in a way its developers did not originally intend.
White hat	Sometimes called an “ethical hacker”—one who uses hacking techniques to assist organizations in improving security.
Black hat	The archetypal hacker—one who uses hacking techniques to break into systems in order to profit, cause chaos, or to satisfy their own goals and interests.
Grey hat	A hacker somewhere in between white hat and black hat; occasionally these hackers will violate laws such as attempting to break into applications without permission, but often for the sake of discovery or recognition rather than profit or to cause chaos.
Penetration tester	Someone who is paid to break into systems, often in the same ways a hacker would. Unlike hackers, penetration testers are paid to report bugs and oversights in the application software so the company that owns the software can fix it before it is broken into by a hacker with malicious intent.
Bug bounty hunter	A freelance penetration tester. Often, large companies will create “responsible disclosure programs” that award cash prizes for reporting security holes. Some bug bounty hunters work full time, but often these are full-time professionals who participate outside of work for extra money.
Application security engineer	Sometimes called a “product security engineer”—a software engineer whose role is to evaluate and improve the security of an organization’s codebase and application architecture.
Software security engineer	A software engineer whose role is to develop security-related products, but who is not necessarily in charge of

Occupation	Description
	evaluating security for the greater organization.
Admin	Sometimes called a “sys admin” or “system administrator.” Admins are technical staff charged with maintaining the configuration and uptime on a web server or web application.
Scrum master	A leadership position in an engineering organization responsible for aiding an engineering team in planning and executing development work.
Security champion	A software engineer not affiliated with a security organization, nor responsible for security work, but interested in improving the security of an organization’s code.

Table P-2. Terms

Term	Description
Vulnerability	A bug in a software system, often as a result of engineering oversight or unexpected functionality when connecting multiple modules together. This particular type of bug allows a hacker to perform unintended actions against the software system.
Threat vector or attack vector	A subsection of application functionality that a hacker deems insecurely written, hence likely to include vulnerabilities and be a good target for hacking.
Attack surface	A list of vulnerabilities in an application that a hacker will build when determining how best to attack a software system.
Exploit	Typically a block of code or list of commands that can be used to take advantage of a vulnerability.
Payload	An exploit that has been formatted in a way that allows it to be sent to a server to take advantage of a vulnerability. Often this just means packaging up an exploit into the proper format to be sent over a network.
Red team	A team often composed of penetration testers, network security engineers, and software security engineers. This team attempts to hack into a company's software to assess the company's ability to stand up against actual hackers.
Blue team	A team often composed of software security engineers and network security engineers. This team attempts to improve a company's software security, often using feedback from a red team to drive prioritization.
Purple team	A team that performs a combination of both red team and blue team role responsibilities. A general-purpose security team rather than a specialized team; often more difficult to correctly staff due to expansive skill requirements.

Term	Description
Website	A series of information documents accessible via the internet, typically over the HTTP protocol.
Web application	A desktop-like application that is delivered via the internet and run inside of a browser rather than a host operating system. These differ from traditional websites in that they have many levels of permissions, store user input in databases, and often allow users to share content with each other.
Hybrid application	A mobile application that is built on top of web-based technology. Typically these make use of another library, like Apache's Cordova, in order to share native functionality with the web application on top.

Table P-3. Acronyms

Acronym	Description
API	Application programming interface—a set of functions exposed by one code module with the intent for other code to consume and make use of it. Typically used in this book when referring to functions exposed over HTTP that a browser can call on a server. Can also be used when referring to modules communicating locally, including separate modules in the same software package.
CSRF	Cross-Site Request Forgery—an attack in which a hacker is able to take advantage of a privileged user's permissions in order to make requests against a server.
CSS	Cascading Style Sheets—a styling language usually used in combination with HTML to create visually appealing and properly aligned UI.
DDoS	Distributed denial of service—a DoS attack that is performed at scale by multiple computers at once, overwhelming a server with sheer numbers; a single computer would likely not be able to cause such mayhem.
DOM	Document Object Model—an API that is shipped with every web browser. Includes all the necessary functionality for organizing and managing the HTML in the page alongside APIs for managing history, cookies, URLs, and other common browser functionality.
DoS	Denial of service—an attack that focuses not on stealing data, but instead on requesting so many server or client resources that the application user experience is worsened or the application no longer functions.
HTML	HyperText Markup Language—a templating language used on the web alongside CSS and JavaScript.
HTTP	HyperText Transfer Protocol—the most commonly used networking protocol for communicating between clients and servers in a web application or website.

Acronym	Description
HTTPS	HyperText Transfer Protocol Secure—HTTP traffic that is encrypted using either HTTP over TLS or HTTP over SSL.
JSON	JavaScript Object Notation—a specification for storing hierarchical data in a way that is lightweight, easy to read by humans, and easy to read by machines. Often used when communicating between the browser and a web server in modern web applications.
OOP	Object-oriented programming—a programming model that organizes code around objects and data structures rather than functionality or logic.
OSS	Open source software—software that is freely available for both consumption and for modification. Often published under licenses like MIT, Apache, GNU, or BSD.
REST	Representational State Transfer—a specific architecture for building stateless APIs that define API endpoints as resources rather than functional units. Many data formats are permitted in REST, but typically JSON is used.
RTC	Real-time communication—a newer networking protocol that allows browsers to communicate with each other and web servers.
SOAP	Simple Object Access Protocol—a protocol for function-driven APIs that require strictly written schemas. Only supports XML as a data format.
SOP	Same Origin Policy—a browser-enforced policy that prevents content from one origin from being loaded in another origin.
SPA	Single-page application—also called “single-page web application” (SPWA). Refers to a website on the internet that functions similarly to a desktop application, managing its own UI and state rather than using the browser-provided defaults.
SSDL	Secure software development life cycle—also called SDLC/SDL. A common framework that allows software engi-

Acronym	Description
	neers and security engineers to work together in order to write more secure code.
SSL	Secure Sockets Layer—a cryptographic protocol designed for securing information in transit (over the network), in particular for use in HTTP.
TLS	Transport Layer Security—a cryptographic protocol designed for securing information in transit (over the network), typically used in HTTP. This protocol replaced SSL, which is now deprecated.
VCS	Version control system—a special type of software used for managing historical additions and redactions from a codebase. Sometimes also includes dependency management and collaboration features.
XML	Extensible Markup Language—a specification for storing hierarchical data that adheres to a strict set of rules. Heavier weight than JSON but more configurable.
XSS	Cross-Site Scripting—a type of attack that involves forcing another client (often a browser) to run code written by a hacker.
XXE	XML External Entity—an attack that relies on an improperly configured XML parser to steal local files on the web server or include malicious files from another web server.

Summary

This is a multifaceted book designed to benefit those with both offensive and defensive security interests. It is easily accessible for any type of developer or administrator with a sufficient web programming background (client and server) to understand and use.

Web Application Security walks you through a number of techniques talented hackers and bug bounty hunters use to break into applications,

then teaches you the techniques and processes you can implement in your own software to protect against such hackers.

This book can be read from cover to cover or as an on-demand reference for particular types of recon techniques, attacks, and defenses against attacks. Ultimately, its aim is to aid you in becoming better at web application security in a way that is practical, hands-on, and follows a logical progression such that no significant prior security experience is required.

I sincerely that hope the hundreds of hours that have gone into writing this book are beneficial to you, and that you derive some interesting learning from its contents. You are welcome to reach out to me with any feedback or suggestions for future editions.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`*Constant width*`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`*Constant width italic*`

Shows text that should be replaced with user-supplied values or by values determined by context.

TIP

This element signifies a tip or suggestion.

NOTE

This element signifies a general note.

WARNING

This element indicates a warning or caution.

O’Reilly Online Learning

NOTE

For more than 40 years, *O’Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O’Reilly’s online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O’Reilly and 200+ other publishers. For more information, visit *https://oreilly-com.ezproxy.sfpl.org*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-889-8969 (in the United States or Canada)

707-829-7019 (international or local)

707-829-0104 (fax)

support@oreilly.com

https://www-oreilly-com.ezproxy.sfpl.org/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/web-application-security-2e*.

For news and information about our books and courses, visit *https://oreilly-com.ezproxy.sfpl.org*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*.

Follow us on Twitter: *https://twitter.com/oreillymedia*.

Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgments

Special thanks to the following people:

- 1st edition editors: Angela Rufino and Jennifer Pollock
- 1st edition technical review: August Detlefsen, Ryan Flood, Chetan Karande, Allan Liska, and Tim Gallo
- 2nd edition editors: Angela Rufino, Katherine Tozer, and Tonya Trybula
- 2nd edition technical review: Nishil Shah, Dustin Kinsey, Caroline Wong, Jon Lamendola, Ming Chow, and Chetan Karande

I also want to give a big shout-out to my wife Amy, who has supported me for over a decade throughout a number of difficult projects, including both editions of this book.

[1] O'Reilly aims to avoid using figurative language that associates positive or negative characteristics with colors that are also associated, problematically, with people. In this book, we use the terms "black hat" and "white hat" when an alternative is not yet used widely enough in information security to be clear.