Chapter 9. Introduction to Hacking Web Applications

In Part II, we will be building on top of our recon skills in order to learn about particular exploits we can use to take advantage of vulnerabilities in web applications. Here you will learn how to take on the role of a hacker.

We will attack the hypothetical web application presented in Part I, *megabank.com*, using a wide array of exploits, all of which are extremely common and found often throughout many of today's web applications. The skills acquired from Part II can easily be migrated elsewhere, as long as you also apply the skills and techniques from Part I. By the end of Part II, you will have both the recon skills required to find bugs in applications that you can exploit and the offensive hacking skills required to build and deploy payloads that take advantage of those security bugs.

The Hacker's Mindset

Becoming a successful hacker takes more than a set of objectively measurable skills and knowledge—it also takes a very particular mindset.

Software engineers measure productivity in value-add through features, or improvements, to an existing codebase. A software engineer might say, "I added features x and y; therefore, today was a good day." Alternatively, they might say, "I improved the performance of features a and b by 10%," alluding to the fact that the work of a software engineer, while difficult to measure compared to traditional occupations, is still quantifiably measurable.

Hackers measure productivity in ways that are much more difficult to discern and measure. This is because the majority of hacking is actually data gathering and analysis. Often this process is riddled with false positives and might look like time wasted to an uneducated onlooker.

Most hackers don't deconstruct or modify software but instead analyze software in order to work with the existing codebase—seeking entrypoints rather than making them. Often the skills used to analyze an application while seeking entrypoints are similar, if not identical, to the skills presented in the first part of this book.

Any given codebase is full of bugs that could potentially be exploitable. A good hacker is constantly on the lookout for clues that could lead to the discovery of a vulnerability.

Unfortunately, the nature of this work means that even a good hacker can go a significant amount of time without a big success. It's entirely possible to spend weeks, if not months, analyzing a web application before a suitable entrypoint can be found and an exploit can be designed and delivered.

As a hacker, you need to constantly reinforce the importance of finding and delivering a payload. Beyond that, you must also carefully keep a record of your prior attempts and the lessons learned from them. Attention to detail when logging prior work will be crucial as you move from exploring small applications and begin hacking larger applications, in particular with key functionality or data as the target.

As we saw in the history of software security, hackers must also constantly be improving their skill set, otherwise they will be bested by those who intend to keep hackers out of their software. This means that a hacker must also be constantly learning, as old techniques may become less valuable as the web adapts.

A hacker is first and foremost a detective. A good hacker is a detective who is properly organized, and a great hacker is a good hacker who happens to also have excellent technical knowledge and skills. A master

hacker has all of the above and is constantly learning and adapting their skill set as those who try to ward them off improve upon their own skills.

Applied Recon

In Part I we learned how to scout a web application, learning various bits about its underlying technology and structure along the way. This part is about taking advantage of security holes in the same applications.

The lessons from Part I are not to be forgotten, however. These lessons will be crucial going forward, and you will soon understand why.

In Part I you learned how to determine what type of API an application is using to serve data to its clients (the browser in our examples). We learned that most modern web applications use REST APIs to accomplish this. The examples in the following chapters will mostly involve sending a payload over a REST API. As a result, being able to determine the API type of an application you are trying to hack will be important here.

Furthermore, we used a combination of public records and network scripts to discover undocumented API endpoints. In this chapter, the exploits we develop will be applicable to many different web applications. As we learned in Part I, sometimes it can be valuable to try the same exploit against multiple applications with the same owner. It's very possible that due to code reuse, you could find an exploit against a single web application and replicate it to internal web applications discovered via the techniques discussed in prior chapters.

The topics surrounding endpoint discovery will likewise be beneficial, as you may encounter multiple API endpoints that take a similarly structured payload. Perhaps an attack against `/users/1234/friends` does not return any sensitive nonpublic data, but `/users/1234/settings` could.

Understanding how to figure out the authentication scheme in place for a web application is also crucial. Most web applications today offer a superset of guest functionality to authenticated users. This means the number of APIs you can attack with an authentication token is greater, and the

privileges given to the processes run as a result of those requests being made will likely be greater.

In Part I we also learned how to identify third-party dependencies (often OSS) in an application. In this part we will learn how to find and customize publicly documented exploits against third-party dependencies. Sometimes we may even find a security hole that resulted from an integration between custom code and third-party code.

Our discussions and analysis surrounding application architecture will be valuable here, as we may find that while application A cannot be exploited, application B can. If we do not have a way of deploying an exploit directly to application B, we may instead look into the ways that application A communicates with application B in order to attempt to find a way to deliver our payload to application A, which would then later communicate it to application B.

To conclude and once again point out, the recon skills of the prior chapters and the hacking skills in the upcoming chapters go hand in hand. Hacking and recon are all complex and interesting skills on their own, but together they are significantly more valuable.