Part II. Offense

In Part I of this book, we explored a number of ways to investigate and document the structure and function of a web application. We evaluated ways of finding APIs on a server, including those that exist on subdomains rather than at just the top-level domain. We considered methods of enumerating the endpoints that those APIs exposed and the HTTP verbs that they accepted.

After building out a map of subdomains, APIs, and HTTP verbs, we looked at ways of determining what type of request and response payloads would be accepted by each endpoint. We approached this from a generic angle, as well as by looking at methods of finding open specifications that would lead us to the payload's structure more rapidly.

After investigating ways of mapping out an application's API structure, we began a conversation regarding third-party dependencies and evaluated various ways of detecting third-party integrations on a first-party application. From this investigation, we learned how to detect SPA frameworks, databases, and web servers, and learned general techniques (like fingerprinting) to identify versions of other dependencies.

Finally, we concluded our conversation regarding recon by discussing architectural flaws that can lead to poorly protected functionality. By evaluating a few common forms of insecure web application architecture, we gained insight into dangers faced by hastily developed web applications.

Now in Part II, we will begin learning common techniques used by hackers to break into modern web applications. It is useful to understand the techniques in Part I before you start Part II.

Many of the attacks presented in the following pages are powerful, and sometimes even easy to deploy, but they will not be applicable to any API endpoint, any HTML form, or any web link. We can take advantage of the recon techniques from Part I when looking for ways to apply the exploits in Part II to a real-life web application. Here we will learn about attacks that stem from insecure API endpoints, insecure web forms in the UI, poorly designed browser standards, improperly configured server-side parsers, and more.

By applying the concepts from Part I, we can find API endpoints and determine if they are written insecurely. We can also evaluate client-side (browser) code to see if it handles DOM manipulation correctly or in an insecure manner. Fingerprinting client-side frameworks can be useful for finding weaknesses in an application's UI, as client-side code is stored locally and easy to evaluate. As you can see, the techniques in this book build on top of one another.

In the next few chapters, you will learn how to take advantage of web applications through a number of powerful and common exploitation techniques. As you learn about these techniques, consider the lessons from the previous part and try to brainstorm how those recon techniques would be useful in helping you find weaknesses in an application where the upcoming exploits you'll learn about could be applied.