Chapter 2. Introduction to Web Application Reconnaissance

Web application reconnaissance refers to the explorative data-gathering phase that generally occurs prior to hacking a web application. Web application reconnaissance is typically performed by hackers, pen testers, or bug bounty hunters, but it can also be an effective way for security engineers to find weakly secured mechanisms in a web application and patch them before a malicious actor finds them. Reconnaissance (recon) skills by themselves do not have significant value, but they become increasingly valuable when coupled with offensive hacking knowledge and defensive security engineering experience.

Information Gathering

We already know that recon is all about building a deep understanding of an application before attempting to hack it. We also know that recon is an essential part of a good hacker's toolkit. But so far, our knowledge regarding recon stops about there. So let's brainstorm some more technical reasons as to why recon is important.

WARNING

Many of the recon techniques presented in the following chapters are useful for mapping applications but also could get your IP flagged, potentially resulting in application bans or even legal action. Most recon techniques should only be performed against applications you own or have written permission to test.

Recon can be accomplished in many ways. Sometimes simply navigating through a web application and taking note of network requests will be all that you need to become intimately familiar with the inner workings of that application. However, it is important to note that not all web applications will have a user interface that allows us to visually explore the application and take note of its functionality.

Most public-facing applications (often business-to-consumer apps like social media) will have a public-facing user interface. However, we should not assume that even in this case we have access to the *entire user interface*. Instead, until we have investigated further we should assume that we have access to a *subset of the user interface*.

Let’s think about this logically for a few minutes. When you go to your local MegaBank and open a new bank account (a checking account for this example), you typically also receive login credentials that allow you to check your account information via the web. Usually your account information is entered manually by a bank employee, often by the bank teller who walked you through the paperwork. This means that at one point or another someone else had access to a web or web-connected application that could create new accounts inside the bank’s databases.

Furthermore, if you call and ask your banker to open a new savings account for you, they will do so. Usually they will do this remotely as long as you are able to provide the correct credentials in order to properly identify yourself. With most major banks, this new savings account will be accessible via the same login information that your checking account already uses.

From this we can gather that someone also had access to an application that allowed them to edit information relevant to your (existing) account in order to connect it with the newly created savings account. It could be the same application that was used to create your checking account, or it could be a different application entirely.

Next, you cannot manually close a bank account online, but you can easily walk into your local branch and ask for your account to be closed. After your request is granted, your account will be closed swiftly, typically within a few hours.

You have access to your bank account to check the balance via a web application—but you can often only use this interface to read the balance. This implies you have read-only access.

Some banks may allow us to pay bills or transfer funds online—but none allow customers to create, modify, or delete our own accounts online. Even with the most advanced digital banking systems, the bank's customer has only a limited subset of write-level access. Bank administrators and trusted staff do, however, have the permissions required to modify, create, and delete accounts. Consider the following table, which describes three different users of a banking application:

User	Type	Permissions
Customer	External	Log in to website. Read account balance via web UI.
Teller	Internal	Create new accounts when provided paperwork from a customer.
Banker	Internal	Modify existing accounts on behalf of customers.

It is not feasible for a large bank to hire developers to manually create database queries for each operation that modifies an account, so logically we can expect that they have written automation to do so even though we (external users) cannot access it. Most applications that automate access to data make use of tiered user permissions structures. We call applications with permissions structured like this *role-based access controlled* (RBAC) applications. Very few applications today use only one level of permissions for all users.

You have probably seen these controls in place in software you have used yourself; for example, invoking a dangerous command on your OS might prompt for *admin* credentials. Alternatively, many social media websites have *moderators* who have a higher permissions level than a standard user but are generally below an admin.

If we walked through a web application's UI by itself, we might never learn of API endpoints that are intended for use by these elevated permissions users (such as admins, moderators, etc.). But with a mastery of web application reconnaissance, we can often find these APIs. We can even build a complex map that details the full permissions of an admin or moderator so that we can compare them to the permissions set for a standard user. Occasionally, we might find glitches that allow nonprivileged users to take advantage of functionality intended only for more privileged users.

Recon skills can also be used to gather information regarding applications we literally don't have access to. This could be a school's internal network or a company's network-accessible file server. We don't need a user interface to learn how an application runs if we are equipped with the proper skills to reverse engineer the structure of an application's APIs and the payloads those APIs accept.

Sometimes as you are doing your reconnaissance, you will actually run into servers or APIs that are not protected at all. Many companies rely on multiple servers, both internal and external. Simply forgetting a single line of network or firewall configuration can lead to an HTTP server being exposed to a public network versus being confined to an internal network.

As you build up a map of what a web application's technology and architecture look like, you will also be able to better prioritize your attacks. You will gain an understanding of what parts of the app are secured the most and which ones could use a bit of work.

Web Application Mapping

As we progress through Part I, you will learn how to build up a map that represents the structure, organization, and functionality of a web application. This should generally be the first step you take before attempting to hack into a web application. As you become more proficient at web application reconnaissance, you will develop your own techniques and your own methods of recording and organizing the information you find.

An organized collection of topographical points is known to many as a *map*. The term *topography* means the study of land features, shapes, and surfaces. Web applications also have features, shapes, and surfaces. These are very different from those you find out in nature, but many of the same concepts hold true. We will use the term "map" here to define the data points collected regarding the code, network structure, and feature set of an application. You will learn how to acquire the data required to fill a map in the next few chapters.

Depending on the complexity of the application you are testing, and the duration you intend to be testing it for, you may be fine with storing your map in simple scratch notes. For more robust applications, or applications you intend to test frequently and over long periods of time, you probably want a more robust solution. How you choose to structure your own maps is ultimately up to you—any format should be sufficient as long as it is easily traversable and capable of storing relevant information and relationships.

Personally, I prefer to use a *JavaScript Object Notation* (JSON)-like format for most of my notes. I find that hierarchical data structures are very frequently found in web applications, and they also allow me to more easily sort and search my notes.

Here is an example of JSON-like recon notes describing a set of API endpoints found in a web application's API server:

```
{
  api_endpoints: {
    sign_up: {
      url: 'mywebsite.com/auth/sign_up',
      method: 'POST',
      shape: {
        username: { type: String, required: true, min: 6, max: 18 },
        password: { type: String, required: true, min: 6: max 32 },
        referralCode: { type: String, required: false, min: 64, max: 64 }
      }
    },
   sign_in: {
     url: 'mywebsite.com/auth/sign_in',
     method: 'POST',
     shape: {
        username: { type: String, required: true, min: 6, max: 18 },
        password: { type: String, required: true, min: 6: max 32 }
     }
   },
   reset_password: {
    url: 'mywebsite.com/auth/reset',
    method: 'POST',
    shape: {
     username: { type: String, required: true, min: 6, max: 18 },
     password: { type: String, required: true, min: 6: max 32 },
     newPassword: { type: String, required: true, min: 6: max 32 }
    }
   }
  },

 features: {
   comments: {},
   uploads: {
    file_sharing: {}
   },
  },

  integrations: {
   oath: {
    twitter: {},
    facebook: {},
    youtube: {}
   }
  }
}
```

Hierarchical note-taking software like Notion, or mind-mapping software applications like XMind, are also fantastic tools to use for recording and organizing what you have learned through your recon attempts. Ultimately you need to find a method that works well for you, keeping you organized, while also being robust enough to scale beyond simple applications when needed.

Summary

Recon techniques are valuable for developing a deep understanding of the technology and structure of a web application and the services that power that web application. In addition to being able to perform recon against a web application, we also must pay careful attention to our findings and document them in a fashion that is organized enough for easy traversal at a later date.

The JSON-like notes presented in this chapter describe a note-taking style I prefer when documenting my recon efforts against a web application. However, the most important aspect of recon note-taking is to preserve relationships and hierarchies while still keeping the notes easy to read and traverse manually.

You must find a style of documentation that works for you and scales from small applications to large applications. If you find an alternative style or format that suits you better, then use that; the content and structure of the notes are much more important than the application or format in which they are stored.