2

OPEN REDIRECT

We'll begin our discussion with *open redirect* vulnerabilities, which occur when a target visits a website and that website sends their browser to a different URL, potentially on a separate domain. Open redirects exploit the trust of a given domain to lure targets to a malicious website. A phishing attack can also accompany a redirect to trick users into believing they're submitting information to a trusted site when, in reality, their information is being sent to a malicious site. When combined with other attacks, open redirects can also enable attackers to distribute malware from the malicious site or to steal OAuth tokens (a topic we'll explore in Chapter 17).

Because open redirects only redirect users, they're sometimes considered low impact and not deserving of a bounty. For example, the Google bug bounty program typically considers open redirects too low risk to reward. The Open Web Application Security Project (OWASP), which is a community that focuses on application security and curates a list of the most critical security flaws in web applications, also removed open redirects from its 2017 list of top 10 vulnerabilities.

Although open redirects are low-impact vulnerabilities, they're great for learning how browsers handle redirects in general. In this chapter, you'll learn how to exploit open redirects and how to identify key parameters, using three bug reports as examples.

How Open Redirects Work

Open redirects occur when a developer mistrusts attacker-controlled input to redirect to another site, usually via a URL parameter, HTML `<meta>` refresh tags, or the DOM window location property.

Many websites intentionally redirect users to other sites by placing a destination URL as a parameter in an original URL. The application uses this parameter to tell the browser to send a `GET` request to the destination URL. For example, suppose Google had the functionality to redirect users to Gmail by visiting the following URL:

```
https://www.google.com/?redirect_to=https://www.gmail.com
```

In this scenario, when you visit this URL, Google receives a `GET` HTTP request and uses the `redirect_to` parameter's value to determine where to redirect your browser. After doing so, Google servers return an HTTP response with a status code instructing the browser to redirect the user. Typically, the status code is 302, but in some cases it could be 301, 303, 307, or 308. These HTTP response codes tell your browser that a page has been found; however, the code also informs the browser to make a `GET` request to the `redirect_to` parameter's value, *https://www.gmail.com/*, which is denoted in the HTTP response's `Location` header. The `Location` header specifies where to redirect `GET` requests.

Now, suppose an attacker changed the original URL to the following:

```
https://www.google.com/?redirect_to=https://www.attacker.com
```

If Google isn't validating that the `redirect_to` parameter is for one of its own legitimate sites where it intends to send visitors, an attacker could substitute the parameter with their own URL. As a result, an HTTP response could instruct your browser to make a `GET` request to *https://www.<attacker>.com/*. After the attacker has you on their malicious site, they could carry out other attacks.

When looking for these vulnerabilities, keep an eye out for URL parameters that include certain names, such as `url=`, `redirect=`, `next=`, and so on, which might denote URLs that users will be redirected to. Also keep in mind that redirect parameters might not always be obviously named; parameters will vary from site to site or even within a site. In some cases, parameters might be labeled with just single characters, such as `r=` or `u=`.

In addition to parameter-based attacks, HTML `<meta>` tags and JavaScript can redirect browsers. HTML `<meta>` tags can tell browsers to

refresh a web page and make a `GET` request to a URL defined in the tag's `content` attribute. Here is what one might look like:

```
<meta http-equiv="refresh" content="0; url=https://www.google.com/">
```

The `content` attribute defines how browsers make an HTTP request in two ways. First, the `content` attribute defines how long the browser waits before making the HTTP request to the URL; in this case, `0` seconds. Secondly, the `content` attribute specifies the URL parameter in the website the browser makes the `GET` request to; in this case, `https://www.google.com`. Attackers can use this redirect behavior in situations where they have the ability to control the `content` attribute of a `<meta>` tag or to inject their own tag via some other vulnerability.

An attacker can also use JavaScript to redirect users by modifying the window's `location` property through the *Document Object Model (DOM)*. The DOM is an API for HTML and XML documents that allows developers to modify the structure, style, and content of a web page. Because the `location` property denotes where a request should be redirected to, browsers will immediately interpret this JavaScript and redirect to the specified URL. An attacker can modify the window's `location` property by using any of the following JavaScript:

```
window.location = https://www.google.com/
window.location.href = https://www.google.com
window.location.replace(https://www.google.com)
```

Typically, opportunities to set the `window.location` value occur only where an attacker can execute JavaScript, either via a cross-site scripting vulnerability or where the website intentionally allows users to define a URL to redirect to, as in the HackerOne interstitial redirect vulnerability detailed later in the chapter on page 15.

When you're searching for open redirect vulnerabilities, you'll usually be monitoring your proxy history for a `GET` request sent to the site you're testing that includes a parameter specifying a URL redirect.

Shopify Theme Install Open Redirect

Difficulty: Low

URL: *https://apps.shopify.com/services/google/themes/preview/supply--blue?domain_name=<anydomain>*

Source: *https://www.hackerone.com/reports/101962/*

Date reported: November 25, 2015

Bounty paid: $500

The first example of an open redirect you'll learn about was found on Shopify, which is a commerce platform that allows people to create stores to sell goods. Shopify allows administrators to customize the look and feel of

their stores by changing their theme. As part of that functionality, Shopify offered a feature to provide a preview for the theme by redirecting the store owners to a URL. The redirect URL was formatted as such:

```
https://app.shopify.com/services/google/themes/preview/supply--blue?domain_name=attacker.com
```

The `domain_name` parameter at the end of the URL redirected to the user's store domain and added `/admin` to the end of the URL. Shopify was expecting that the `domain_name` would always be a user's store and wasn't validating its value as part of the Shopify domain. As a result, an attacker could exploit the parameter to redirect a target to *http://<attacker>.com/admin/* where the malicious attacker could carry out other attacks.

Takeaways

Not all vulnerabilities are complex. For this open redirect, simply changing the `domain_name` parameter to an external site would redirect the user offsite from Shopify.

Shopify Login Open Redirect

Difficulty: Low

URL: *http://mystore.myshopify.com/account/login/*

Source: *https://www.hackerone.com/reports/103772/*

Date reported: December 6, 2015

Bounty paid: $500

This second example of an open redirect is similar to the first Shopify example except in this case, Shopify's parameter isn't redirecting the user to the domain specified by the URL parameter; instead, the open redirect tacks the parameter's value onto the end of a Shopify subdomain. Normally, this functionality would be used to redirect a user to a specific page on a given store. However, attackers can still manipulate these URLs into redirecting the browser away from Shopify's subdomain and to an attacker's website by adding characters to change the meaning of the URL.

In this bug, after the user logged into Shopify, Shopify used the parameter `checkout_url` to redirect the user. For example, let's say a target visited this URL:

```
http://mystore.myshopify.com/account/login?checkout_url=.attacker.com
```

They would have been redirected to the URL *http://mystore.myshopify.com.<attacker>.com/*, which isn't a Shopify domain.

Because the URL ends in *.<attacker>.com* and DNS lookups use the rightmost domain label, the redirect goes to the *<attacker>.com* domain. So when *http://mystore.myshopify.com.<attacker>.com/* is submitted for DNS lookup, it will match on *<attacker>.com*, which Shopify doesn't own, and not *myshopify.com* as

Shopify would have intended. Although an attacker wouldn't be able to freely send a target anywhere, they could send a user to another domain by adding special characters, such as a period, to the values they can manipulate.

Takeaways

If you can only control a portion of the final URL used by a site, adding special URL characters might change the meaning of the URL and redirect a user to another domain. Let's say you can only control the `checkout_url` parameter value, and you also notice that the parameter is being combined with a hardcoded URL on the backend of the site, such as the store URL *http://mystore.myshopify.com/*. Try adding special URL characters, like a period or the @ symbol, to test whether you can control the redirected location.

HackerOne Interstitial Redirect

Difficulty: Low

URL: N/A

Source: *https://www.hackerone.com/reports/111968/*

Date reported: January 20, 2016

Bounty paid: $500

Some websites try to protect against open redirect vulnerabilities by implementing *interstitial web pages*, which display before the expected content. Any time you redirect a user to a URL, you can show an interstitial web page with a message explaining to the user that they're leaving the domain they're on. As a result, if the redirect page shows a fake login or tries to pretend to be the trusted domain, the user will know that they're being redirected. This is the approach HackerOne takes when following most URLs off its site; for example, when following links in submitted reports.

Although you can use interstitial web pages to avoid redirect vulnerabilities, complications in the way sites interact with one another can lead to compromised links. HackerOne uses Zendesk, a customer service support ticketing system, for its *https://support.hackerone.com/* subdomain. Previously, when you followed *hackerone.com* with */zendesk_session*, the browser redirected from HackerOne's platform to HackerOne's Zendesk platform without an interstitial page because URLs containing the *hackerone.com* domain were trusted links. (HackerOne now redirects *https://support.hackerone.com* to *docs.hackerone.com* unless you are submitting a support request via the URL */hc/en-us/requests/new*.) However, anyone could create custom Zendesk accounts and pass them to the `/redirect_to_account?state=` parameter. The custom Zendesk account could then redirect to another website not owned by Zendesk or HackerOne. Because Zendesk allowed for redirecting between accounts without interstitial pages, the user could be taken to the untrusted site without warning. As a solution, HackerOne identified links containing `zendesk_session` as external links, thereby rendering an interstitial warning page when clicked.

In order to confirm this vulnerability, the hacker Mahmoud Jamal created an account on Zendesk with the subdomain *http://compayn.zendesk.com*. He then added the following JavaScript code to the header file using the Zendesk theme editor, which allows administrators to customize their Zendesk site's look and feel:

```
<script>document.location.href = «http://evil.com»;</script>
```

Using this JavaScript, Jamal instructed the browser to visit *http://evil.com*. The `<script>` tag denotes code in HTML and `document` refers to the entire HTML document that Zendesk returns, which is the information for the web page. The dots and names following `document` are its properties. Properties hold information and values that either describe an object or can be manipulated to change the object. So you can use the `location` property to control the web page your browser displays and use the `href` subproperty (which is a property of the `location`) to redirect the browser to the defined website. Visiting the following link redirected targets to Jamal's Zendesk subdomain, which made the target's browser run Jamal's script and redirected them to *http://evil.com*:

```
https://hackerone.com/zendesk_session?locale_id=1&return_to=https://support.hackerone.com/
ping/redirect_to_account?state=compayn:/
```

Because the link includes the domain *hackerone.com*, the interstitial web page doesn't display, and the user wouldn't know the page they were visiting is unsafe. Interestingly, Jamal originally reported the missing interstitial page redirect issue to Zendesk, but it was disregarded and not marked as a vulnerability. Naturally, he kept digging to see how the missing interstitial could be exploited. Eventually, he found the JavaScript redirect attack that convinced HackerOne to pay him a bounty.

Takeaways

As you search for vulnerabilities, note the services a site uses because each represents new attack vectors. This HackerOne vulnerability was made possible by combining HackerOne's use of Zendesk and the known redirect HackerOne was permitting.

Additionally, as you find bugs, there will be times when the security implications aren't readily understood by the person reading and responding to your report. For this reason, I'll discuss vulnerability reports in Chapter 19, which details the findings you should include in a report, how to build relationships with companies, and other information. If you do some work up front and respectfully explain the security implications in your report, your efforts will help ensure a smoother resolution.

That said, there will be times when companies don't agree with you. If that's the case, continue to dig like Jamal did and see if you can prove the exploit or combine it with another vulnerability to demonstrate impact.

Summary

Open redirects allow a malicious attacker to redirect people unknowingly to a malicious website. Finding them, as you learned from the example bug reports, often requires keen observation. Redirect parameters are sometimes easy to spot when they have names like `redirect_to=`, `domain_name=`, or `checkout_url=`, as mentioned in the examples. Other times, they might have less obvious names, such as `r=`, `u=`, and so on.

The open redirect vulnerability relies on an abuse of trust where targets are tricked into visiting an attacker's site while thinking they're visiting a site they recognize. When you spot likely vulnerable parameters, be sure to test them thoroughly and add special characters, like a period, if some part of the URL is hardcoded.

The HackerOne interstitial redirect shows the importance of recognizing the tools and services websites use while you hunt for vulnerabilities. Keep in mind that you'll sometimes need to be persistent and clearly demonstrate a vulnerability to persuade a company to accept your findings and pay a bounty.