XSS Workshop

Evan King, esk79

September 2016

1 What is Cross Site Scripting?

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantec as of 2007. [1] There are three types of XSS: reflected, stored, and DOM based. We will look at all three in this workshop.

This vulnerability can be used to conduct a number of browser-based attacks including [2]:

1. Hijacking another user's browser
2. Capturing sensitive information viewed by application users
3. Pseudo defacement of the application
4. Port scanning of internal hosts ("internal" in relation to the users of the web application)
5. Directed delivery of browser-based exploits
6. Other malicious activities

2 The payload

The goal of XSS is most often to steal the cookies of other users and, ideally, the application's admins. Using the stolen session cookie, the hacker is then able to successfully impersonate the compromised user. The most common payload to check if one can inject client-side scripts into web page is the following:

```
// Most common test payload
<script>alert(1);</script>
```

If this payload is successful, you will see an alert box pop up in your browser. If such pop up appears, then the application is indeed vulnerable to XSS and you can begin crafting your malicious payload.

As mentioned above, the most common payload steals other users session cookies. We will not be constructing this payload for you (that would ruin the fun), but we will give you a hint. The following Javascript returns the users current cookies.

```
// Javascript that returns users cookies
document.cookie
//output
''Session_id=uwNYXFyxC3d8NA3--jHa_3u8OFC1bsXA''
```

3 Reflected XSS

Reflected XSS if the most common form of XSS. It occur when an attacker injects browser executable code within a single HTTP response. The injected attack is not stored within the application itself; it is non-persistent and only impacts users who open a maliciously crafted link or third-party web page. The attack string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.[2]

3.1 The Goal

The website we've provided at youcanthack.me:1234 has a reflected XSS vulnerability, can you find it?

3.2 Hack Steps

1. Choose a unique arbitrary string that does not appear anywhere within the application and that contains only alphabetical characters and therefore is unlikely to be affected by any XSS-specific filters. For example:

   myxsstestdmqlwp

   Submit this string as every parameter to every page, targeting only one parameter at a time.

2. Monitor the application's responses for any appearance of this same string. Make a note of every parameter whose value is being copied into the application's response. These are not necessarily vulnerable, but each instance identified is a candidate for further investigation.

3. Review the HTML source to identify the location(s) where your unique string is being reflected.

4. If the string appears more than once, each occurrence needs to be treated as a separate potential vulnerability and investigated individually.

5. Determine, from the location within the HTML of the user-controllable string, how you need to modify it to cause execution of arbitrary script. Typically, numerous different methods will be potential vehicles for an attack. This is a trial and error process.

6. Test your exploit by submitting it to the application. If your crafted string is still returned unmodified in the source code and **is not escaped**, the application is vulnerable. Double-check that your syntax is correct by using a proof-of-concept script to display an alert dialog, and confirm that this actually appears in your browser when the response is rendered.

7. Now you can construct your malicious payload and send your infected url to your victim.

3.3 Hints

1. Always check the url when looking for places in the application that accepts user input. Do not limit yourself to only input text boxes, developers often know to fully sanitize user input through this means.

2. Be aware of url encoding. Before typing into a url, make sure that your payload is properly encoded.You can use an on-line tool like this one

3. You do not need to log in in order to find this vulnerability.

3.4 You did it!

Congratulations! Assuming you successfully found the vulnerability and developed your payload, you are now ready to attack another user. To do this, you would send the “infected” url that has your payload to a victim, waiting for them to click on the link and, thus, triggering the payload. Of course, this is just a simulation so we will do no such thing.

4 DOM Based XSS

DOM Based XSS (or as it is called in some texts, “type-0 XSS”) is an XSS attack wherein the attack payload is executed as a result of modifying the DOM “environment” in the victim’s browser used by the original client side script, so that the client side code runs in an “unexpected” manner. That is, the page itself (the HTTP response that is) does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment.

This is in contrast to other XSS attacks (stored or reflected), wherein the attack payload is placed in the response page (due to a server side flaw).[4]

4.1 The Goal

The website we've provided at youcanthack.me:1234 has a DOM based XSS vulnerability, can you find it? This is the easiest of the 3 to find on the website.

4.2 Hack Steps

1. Review every piece of client-side JavaScript for the following APIs, which may be used to access DOM data that can be controlled via a crafted URL:

   (a) document.location

   (b) document.URL

   (c) document.URLUnencoded

   (d) document.referrer

   (e) window.location

2. In every instance where one of the preceding APIs is being used, closely review the code to identify what is being done with the user-controllable data, and whether crafted input could be used to cause execution of arbitrary JavaScript. In particular, review and test any instance where your data is being passed to any of the following APIs:

   (a) document.write()

   (b) document.writeln()

   (c) document.body.innerHtml

   (d) eval()

   (e) window.execScript()

   (f) window.setInterval()

   (g) window.setTimeout()

4.3 Hints

1. BIG HINT! Look at the "Tools" page on the website, specifically at the Line Memory Calculator tool that we've built.

2. Remember, the goal of DOM Based and Reflected XSS is to construct a malicious url that you can send to an unsuspecting user. If you do not have such a url, you do not have a vulnerability.

5 Stored XSS

Stored Cross-site Scripting (XSS) is the most dangerous type of Cross Site Scripting. Web applications that allow users to store data are potentially exposed to this type of attack. Stored XSS occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use, typically in a database. The input that is stored is not correctly filtered. As a consequence, the malicious data will appear to be part of the web site and run within the user's browser under the privileges of the web application.[2]

5.1 The Goal

The website we've provided at youcanthack.me:1234 has a stored XSS vulnerability, can you not only find it, but exploit it in such a way that you are able to login to the website as the admin?

5.2 Hack Steps

1. Typically, but not always, stored XSS vulnerabilities require a user to be logged in. Find the login form and create a fake user in order to access user restricted portions on the website.

2. Having submitted a unique string to every possible location within the application, you must review all of the application's content and functionality once more to identify any instances where this string is displayed back to the browser. User-controllable data entered in one location (for example, a name field on a personal information page) may be displayed in numerous places throughout the application. (For example, it could be on the user's home page, in a listing of registered users, in work flow items such as tasks, on other users' contact lists, in messages or questions posted by the user, or in application logs.) Each appearance of the string may be subject to different protective filters and therefore needs to be investigated separately.

3. If possible, all areas of the application accessible by administrators should be reviewed to identify the appearance of any data controllable by non-administrative users. For example, the application may allow administrators to review log files in-browser. It is extremely common for this type of functionality to contain XSS vulnerabilities that an attacker can exploit by generating log entries containing malicious HTML.

4. If the application allows files to be uploaded and downloaded, always probe this functionality for stored XSS attacks. Detailed techniques for testing this type of functionality are discussed later in this chapter.

5. Think imaginatively about any other possible means by which data you control may be stored by the application and displayed to other users. For

example, if the application search function shows a list of popular search items, you may be able to introduce a stored XSS payload by searching for it numerous times, even though the primary search functionality itself handles your input safely.

6. Again, test your exploit by submitting it to the application. If your crafted string is still returned unmodified in the source code and **is not escaped**, the application is vulnerable. Double-check that your syntax is correct by using a proof-of-concept script to display an alert dialog, and confirm that this actually appears in your browser when the response is rendered.

7. In order for you to reap the results of your malicious script you will need to have a server running that is logging all requests to that server. This is easy enough to do, however, we have done it for you. You will see the server logs on the projector. Place your netid in your payload so that if it is successful you will be able to see your netid on the projector, thus, confirming your success. The server that we have set up can be found out youcanthack.me:1000.

5.3 Hints

1. If you're unsure about how to construct the payload, do some Googling. You want to somehow make a request to your own server that you set up (for this example the one we set up for you), which has the session cookie in the request.

2. Once you have the session cookie, you've done all of the hard work! Now you just need to change your cookie to the one that you captured. There are lots of ways to do this. If you are a chrome user, check out the plug-in EditThisCookie.

6 Sources

[1] https://en.wikipedia.org/wiki/Cross-site_scripting
[2] https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting
[3] The Web Application Hacker Handbook by Dafydd Stuttard and Marcus Pinto
[4] https://www.owasp.org/index.php/DOM_Based_XSS