XSS Discovery

By now, we should have a good understanding of what an XSS vulnerability is, the three types of XSS, and how each type differs from the others. We should also understand how XSS works through injecting JavaScript code into the client-side page source, thus executing additional code, which we will later learn how to utilize to our advantage.

In this section, we will go through various ways of detecting XSS vulnerabilities within a web application. In web application vulnerabilities (and all vulnerabilities in general), detecting them can become as difficult as exploiting them. However, as XSS vulnerabilities are widespread, many tools can help us in detecting and identifying them.

Automated Discovery

Almost all Web Application Vulnerability Scanners (like Nessus, Burp Pro, or ZAP) have various capabilities for detecting all three types of XSS vulnerabilities. These scanners usually do two types of scanning: A Passive Scan, which reviews client-side code for potential DOM-based vulnerabilities, and an Active Scan, which sends various types of payloads to attempt to trigger an XSS through payload injection in the page source.

While paid tools usually have a higher level of accuracy in detecting XSS vulnerabilities (especially when security bypasses are required), we can still find open-source tools that can assist us in identifying potential XSS vulnerabilities. Such tools usually work by identifying input fields in web pages, sending various types of XSS payloads, and then comparing the rendered page source to see if the same payload can be found in it, which may indicate a successful XSS injection. Still, this will not always be accurate, as sometimes, even if the same payload was injected, it might not lead to a successful execution due to various reasons, so we must always manually verify the XSS injection.

Some of the common open-source tools that can assist us in XSS discovery are XSS Strike, Brute XSS, and XSSer. We can try `XSS Strike` by cloning it to our VM with `git clone`:

XSS Discovery

```
MisaelMacias@htb[/htb]$ git clone https://github.com/s0md3v/XSStrike.git
MisaelMacias@htb[/htb]$ cd XSStrike
MisaelMacias@htb[/htb]$ pip install -r requirements.txt
MisaelMacias@htb[/htb]$ python xsstrike.py

XSStrike v3.1.4
...SNIP...
```

We can then run the script and provide it a URL with a parameter using `-u`. Let's try using it with our `Reflected XSS` example from the earlier section:

XSS Discovery

```
MisaelMacias@htb[/htb]$ python xsstrike.py -u "http://SERVER_IP:PORT/index.php?task=test"

        XSStrike v3.1.4

[~] Checking for DOM vulnerabilities
[+] WAF Status: Offline
[!] Testing parameter: task
[!] Reflections found: 1
[~] Analysing reflections
[~] Generating payloads
[!] Payloads generated: 3072
------------------------------------------------------------
[+] Payload: <HtMl%09onPoIntERENTER+=+confirm()>
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N]
```

As we can see, the tool identified the parameter as vulnerable to XSS from the first payload. `Try to verify the above payload by testing it on one of the previous exercises. You may also try testing out the other tools and run them on the same exercises to see how capable they are in detecting XSS vulnerabilities.`

Manual Discovery

When it comes to manual XSS discovery, the difficulty of finding the XSS vulnerability depends on the level of security of the web application. Basic XSS vulnerabilities can usually be found through testing various XSS payloads, but identifying advanced XSS vulnerabilities requires advanced code review skills.

XSS Payloads

The most basic method of looking for XSS vulnerabilities is manually testing various XSS payloads against an input field in a given web page. We can find huge lists of XSS payloads online, like the one on PayloadAllTheThings or the one in PayloadBox. We can then begin testing these payloads one by one by copying each one and adding it in our form, and seeing whether an alert box pops up.

> Note: XSS can be injected into any input in the HTML page, which is not exclusive to HTML input fields, but may also be in HTTP headers like the Cookie or User-Agent (i.e., when their values are displayed on the page).

You will notice that the majority of the above payloads do not work with our example web applications, even though we are dealing with the most basic type of XSS vulnerabilities. This is because these payloads are written for a wide variety of injection points (like injecting after a single quote) or are designed to evade certain security measures (like sanitization filters). Furthermore, such payloads utilize a variety of injection vectors to execute JavaScript code, like basic `<script>` tags, other `HTML Attributes` like `<img>`, or even `CSS Style` attributes. This is why we can expect that many of these payloads will not work in all test cases, as they are designed to work with certain types of injections.

This is why it is not very efficient to resort to manually copying/pasting XSS payloads, as even if a web application is vulnerable, it may take us a

Cheat Sheet

Go to Questions

Table of Contents

XSS Basics
Intro to XSS ✓
Stored XSS ✓
Reflected XSS ✓
DOM XSS ✓
XSS Discovery ✓
XSS Attacks
Defacing ✓
Phishing ✓
Session Hijacking ✓
XSS Prevention
XSS Prevention ✓
Skills Assessment
Skills Assessment ✓

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

while to identify the vulnerability, especially if we have many input fields to test. This is why it may be more efficient to write our own Python script to automate sending these payloads and then comparing the page source to see how our payloads were rendered. This can help us in advanced cases where XSS tools cannot easily send and compare payloads. This way, we would have the advantage of customizing our tool to our target web application. However, this is an advanced approach to XSS discovery, and it is not part of the scope of this module.

Code Review

The most reliable method of detecting XSS vulnerabilities is manual code review, which should cover both back-end and front-end code. If we understand precisely how our input is being handled all the way until it reaches the web browser, we can write a custom payload that should work with high confidence.

In the previous section, we looked at a basic example of HTML code review when discussing the `Source` and `Sink` for DOM-based XSS vulnerabilities. This gave us a quick look at how front-end code review works in identifying XSS vulnerabilities, although on a very basic front-end example.

We are unlikely to find any XSS vulnerabilities through payload lists or XSS tools for the more common web applications. This is because the developers of such web applications likely run their application through vulnerability assessment tools and then patch any identified vulnerabilities before release. For such cases, manual code review may reveal undetected XSS vulnerabilities, which may survive public releases of common web applications. These are also advanced techniques that are out of the scope of this module. Still, if you are interested in learning them, the Secure Coding 101: JavaScript and the Whitebox Pentesting 101: Command Injection modules thoroughly cover this topic.

Connect to Pwnbox
Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location
UK 139ms

Terminate Pwnbox to switch location

Start Instance
∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Cheat Sheet

Answer the question(s) below to complete this Section and earn cubes!

Target(s): Click here to spawn the target system!

+2 Utilize some of the techniques mentioned in this section to identify the vulnerable input parameter found in the above server. What is the name of the vulnerable parameter?

email

Submit Hint

+2 What type of XSS was found on the above server? "name only"

reflected

Submit Hint

Previous Next Mark Complete & Next

Powered by HACKTHEBOX