# Assignment 6

New Attempt

- Due Oct 31, 2024 by 11:59pm
- Points 100
- Submitting a file upload
- File Types pdf

Due: Tuesday, October 31, 2024 (11:59 PM). Please submit your assignment on Canvas as a PDF.

# Cross-site Scripting and JavaScript Injection

Cross-site Scripting (XSS) is a client-side script injection attack. Attacker tries to execute malicious code in a Web browser of the victim by injecting it into a Web page. The attack is activated when the victim visits that Web page and executes the malicious script. Vulnerable web pages that can deliver malicious scripts are forums, message boards and various web pages that accept input from users. Web pages may become vulnerable to XSS when they allow unsanitized input. XSS attacks can be executed in VBScript ActiveX, Flash, JavasScript, or CSS. However, the most common way is using JavasScript.

In this assignment, you will get familiar with browser attacks. One of the common tools used in these types of attacks is BeEF (The Browser Exploitation Framework Project) which we learn how to use in this assignment.

# JavaScript Injection

JavaScript Injection is the process in which one can insert malicious JavaScript code into a victim machine and run it. This process can be performed by either finding an XSS vulnerability on a website or by performing a man in the middle attack and injecting HTML files with a malicious script. In this assignment, you experience both types of JavaScript Injection.

# BeEF

**BeEF (https://beefproject.com/)** is a penetration testing tool that focuses on Web browsers. BeEF works in a very simple way. There is a JavaScript file called hook.js that should be loaded to a target browser. Then, this remote code can be executed in the target browser which is able to do a number of tasks such as opening new windows, loading pages, and etc. The default location of hook.js is `http://<Kali's IP>:3000/hook.js`.

# Setup

For this assignment, you will need Kali and Windows 7 VMS (as well as Router for obvious reasons).

Ensure that the certificate for mitmproxy used in any previous assignment is uninstalled.

You may need to install it again in part 3, but we assume that it does not exist in the victim machine until part 3.

You need to have docker and docker-compose installed on your Kali machine to run the backend and frontend servers.

In order to start BeEF click Applications > Exploitation Tools > BeEF

You will be asked to setup a password and will be redirected to the BeEF website.

The recommended browser for this assignment is Google Chrome.

# 1 JavaScript Injection & BeEF Introduction

For this set of tasks, we have created a website in order for you to perform XSS JavaScript Injection. This website contains a search page for Task1 and message board for Task2. You can download it from **here (https://canvas.sfu.ca/courses/85026/files/24007558?wrap=1)** ↓ **(https://canvas.sfu.ca/courses/85026/files/24007558/download?download_frd=1)** . Download the Zip file and extract it. Then, open a terminal, and navigate to ../XSS.

In order to run the website on Kali, run

```
docker compose up
```

This will build the frontend and backend docker images and start both the frontend and backend serves on http://localhost:5050 and ttp://localhost:3000.

Note: The frontend is an actual website that the victim is trying to visit. The backend is an attacker-controlled website. We're running both the backend and frontend on the Kali machine to reduce the complexity of the setup.

In the victim machine visit `http://<Kali's IP>:3000/task1?q=42` preferably from a Chrome browser. As you can see the search term is equal to q's value in the URL. This is the most common way to pass parameters in the URL. The most straightforward approach to perform a JavaScript injection is entering a JavaScript code in the URL query parameter or in a form textbox.

**Task 1 (2 %)**: Perform a JavaScript injection by changing the URL query parameter (parameter "q") in `http://<Kali's IP>:3000//task1?q=42`. The code should raise a popup (alert). Report the URL you created for this purpose.

**Task 2 (5 %)**: Visit `http://<Kali's IP>:3000/task2`. This time this is a message board. Create an alert() JavaScript code and inject it in the message board. Report the message you entered to confirm that you get an alert.

**Task 3 (10 %)**: In this task, you perform cookie stealing. Suppose that the victim (Windows 7) has access to `http://<Kali's IP>:3000//task1?q=42`. And you have a server running on the attacker listening on port 5050 (http://<Kali's IP>:5050) that logs all the requests made to it. What is the URL you can send to the victim in order to steal their cookies and send them to the server listening on Kali on port 5050? Note that the server can log all the request query parameters.

*Hint*: There may be different ways to solve

**Task 3.1** If what you are trying to do is not working, report your rationale for partial credit. Also, JavaScriptcode to create an alert showing all the cookies is <script>alert(document.cookie)</script>

For the rest of the tasks, you are going to use BeEF.

BeEF is already installed on the system at /usr/share/beef-xss. If this version has some issues, you can install BeEF from the source using these steps:

1. git clone **https://github.com/beefproject/beef.git (https://github.com/beefproject/beef.git)**
2. cd beef
3. ./install

Then start beef using ./beef (Note that you may need to change the default username/password in config.yaml)

**Task 4 (4 %)**: Assuming you have already started BeEF, perform the simplest way to hook by navigating in Kali's browser to **http://127.0.0.1:3000/demos/basic.html (http://127.0.0.1:3000/demos/basic.html)** . This will load the `hook.js` . After that confirm that your browser is hooked by checking Online Browsers in BeEF's left panel. After doing so close the demo page you opened and wait a bit to confirm that now there is no browser in Online Browsers.

Now do the same from the target browser (Windows 7 Chrome) but instead of navigating to **http://127.0.0.1:3000/demos/basic.html (http://127.0.0.1:3000/demos/basic.html)** , navigate to http://<Kali's IP>:3000/demos/basic.html Ensure that the browser is hooked.

Post a screenshot of the hooked browser in Windows 7, and report the list of browser's plugins through BeEF (*Hint*: Search Browser info through BeEF).

Now start the Apache2 Web Server:

```
service apache2 start
cd /var/www/html
mv index.html index.html.old
```

**Task 5 (2 %)**: Create a sample Web page that is infected with hook.js (**http://localhost:3000/hook.js (http://localhost:3000/hook.js)** ). Store the website in a file called `index.html` in path `/var/www/html` . Visit the Web page from the target machine and confirm that the browser is hooked. Report the contents of the index.html file.

## 2 Using Bettercap for Javascript Injection

Although the approach you used to hook the browser in Task5 works fine, it is not realistic to work in real-world attack scenarios. In this task we will perform a man-in-the-middle attack in order to inject a harmless JavaScript file in the target Browser for HTTP requests.

**Task 6 (2 %)**: Open bettercap and perform a man-in-the middle attack against the victim machine. Use also the net sniffer with the verbose flag set to false. Report the commands in bettercap you used to perform the MITM attack.

**Task 7 (10 %)**: In order to inject the script we are going to use an injector script (download it from here (https://drive.google.com/open?id=1HSmGhrkmtRIJsnooMwdVkyYsSBXV8vqY) or here (https://canvas.sfu.ca/courses/85026/files/24007559?wrap=1) (https://canvas.sfu.ca/courses/85026/files/24007559/download?download_frd=1) ). As you can see `line 12` is commented out. Replace the commented out line so that the injected javascript displays a popup (alert) that says “Successful Injection”. Report the complete injector.js you used.

Now in order to inject the Javascript code you are going to use bettercap’s `http.proxy`.

After saving the file in Kali you can perform the Javascript injection by using the following commands (in bettercap)

```
set http.proxy.script /path/to/injector.js
http.proxy on
https.proxy on
```

**Task 8 (2 %)** Navigate to an **HTTP** website (e.g http://solanaceaesource.org/ (http://solanaceaesource.org/) ) and check that the popup window is displayed properly. Post a screenshot of the alert on the HTTP website you visited.

**Task 9 (10 %)**: Now try visiting an HTTPS website. You will not see the alert this time. Why doesn’t this method work on HTTPS websites according to your opinion? How would one be able to make this method work on HTTPS theoretically?

**Task 10 (10 %)**: How does the HTTP Javascript injection work in this case in terms of file/packet inspection? Explain in a few sentences.

## 3 Leveraging the Javascript Injection

Now we will leverage the script injection method by using the `https.proxy module of bettercap`. We assume that the target Windows 7 system has the **mitmproxy certificate** installed (from Assignment 5). If not, perform the necessary steps needed in order to install it (guidelines in Assignment 5).

We will use the key and the certificate produced by mitmproxy in bettercap’s HTTPS proxy to perform HTTPS Javascript injection. You can access the produced certificate and key in `/root/.mitmproxy/mitmproxy-ca-cert.pem` and `/root/.mitmproxy/mitmproxy-ca.pem`, respectively.

**Task 11 (5 %)**: Load the key and the certificate to bettercap’s `https.proxy module`. Then, load the script injector to `https.proxy` module and start the module. Report the commands you used.

**Task 12 (2 %)**: After starting the module go to https://facebook.com (https://facebook.com/) and verify that the injector works. Post screenshot of the website after successful script injection.

## 4 Leveraging JavaScript Injection to a BeEF Hook

In this task, we will leverage the harmless script to the BeEF hook. Remember the hook is at `http://<Kali’s IP>:3000/hook.js`.

**Task 13 (4 %)**: Modify the injector.js to include the hook and remove the alert. Report the modified injector.js.

Please note you need to stop HTTPS/HTTP proxy and reload it in order to pick any changes to the injector. Execute

```
http.proxy off
https.proxy off
http.proxy on
https.proxy on
```

**Task 14 (10 %)**: Now close all tabs and navigate to the HTTPS website **https://twitter.com (https://twitter.com/)** . Go back to BeEF panel and check if the target browser is hooked. It should not be hooked. Now check the certificate of the website you are using. Is it the same as the produced certificate by mitmproxy? If not ensure that it is. Why isn't the injection working?

**Task 15 (10 %)**: Now visit the HTTP website **http://solanaceaesource.org/ (http://solanaceaesource.org/)** . In this case, the injection should work and you should be able to see the web browser in BeEF's **Online Browsers**. Why is this method working, contrary to what you experienced in **Task 14**?

## 5 More BeEF

This task is about using BeEF after a successful JavaScript injection and browser hooking. Assuming that the target browser is hooked, you can navigate to the commands tab and explore the variety of the commands available.

**Task 16 (2 %)**: In BeEF you will find a command which scans for the antivirus in the target system. Report the result of the antivirus scan from BeEF framework and how to perform it. What is the antivirus that the target machine is using?

Following, you will try to use the social engineering module of BeEF and try to gain access to the target system.

**Task 17 (10 %)**: By using the module found in `Social Engineering > Fake Flash Update` explain the steps and the exact commands that a potential attacker can use to gain access to the system and how he/she can make this attack persistent not to lose access even after the victim system reboot. Report the screenshot of how it looks on the victim browser after executing the social engineering attack from BeEF.

## Additional Resources (**No tasks here**)

The below resources are very useful and interesting if you want to dive deeper into XSS or web security in general.

- Portswigger web security: **https://portswigger.net/web-security/all-topics (https://portswigger.net/web-security/all-topics)**
- OWASP Cheat Sheet Series **https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_S**

(https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

- OWASP Testing Guide https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/ (https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/)