

# Lab 12

Venkata Diwakar Reddy Kashireddy

**Abstract**—In this lab, we explored how to perform different types of Cross Site Scripting(XSS) attacks. Cross-Site Scripting (also commonly known as XSS) is a vulnerability/ flaw that combines the allowance of html/script tags as input that are rendered into a browser without encoding or sanitization.

## I. INTRODUCTION

Cross-Site Scripting (XSS) is a major cybersecurity threat where attackers inject malicious scripts into web pages viewed by other users. In this lab, we explore the types, impact, and defense strategies against XSS. It emphasizes the importance of understanding XSS for web security, highlighting how it compromises user data and website integrity. The goal is to provide insights into effective XSS mitigation to enhance overall web safety.

## II. TOOLS

- KVM (Kernel-based Virtual Machine): It is a full virtualization solution for Linux. [1]
- WebGoat: A deliberately insecure web application maintained by OWASP designed for teaching web application security concepts. [2]
- Overleaf: It is a collaborative cloud-based LaTeX editor that helps to create documents easily by providing standard formats. [3]
- GitHub: GitHub is an Internet hosting service for software development and version control using Git. [4]
- Red: – IU Research Desktop (RED) is a virtual desktop service for users with accounts on the Carbonate research supercomputer at IU. [5]
- ZAP: OWASP ZAP (Zed Attack Proxy) is an open-source web application security scanner. It is used by those new to application security and professional penetration testers. [6]
- Firefox DevTools: Firefox Developer Tools is a set of web developer tools built into Firefox. It can be accessed to inspect the web page. [7]

## III. XSS

For the first task, the goal was to check if the cookies of the webgoat site were same, if we opened them in two different tabs. I had used developer tools to get the cookies of each tabs and compared it manually, upon which I found out that, they both were same.

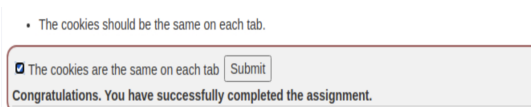


Fig. 1. XSS- checking cookies of webgoat in different tabs

## IV. REFLECTED XSS

This task was to identify which field was vulnerable to XSS, I employed a straightforward strategy. As we are aware that XSS often occurs when user input isn't properly sanitised, I focused on areas where user input would be echoed back. My reasoning was that fields accepting text inputs, as opposed to numerical ones, were more likely to be susceptible. I used the

```
<script>alert(1)</script>
```

command, for detecting XSS vulnerabilities. This script was entered into various input fields, particularly the 'Enter your credit card number' field during a simulated purchase. This field, expected to display input in a purchase confirmation.

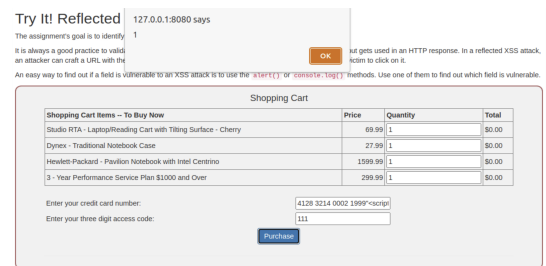


Fig. 2. Checking XSS vulnerable field

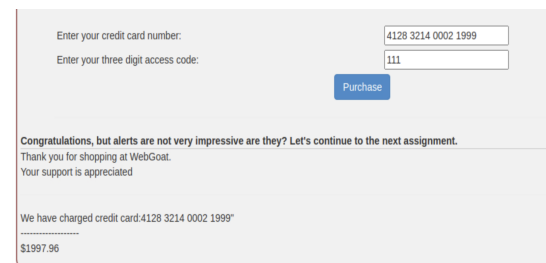


Fig. 3. Task completion confirmation

## V. IDENTIFY POTENTIAL FOR DOM-BASED XSS

For the next task, the aim was to find a DOM-Based XSS vulnerability in WebGoat, focusing on the client-side route configurations. Using the browser's developer tools, I investigated the WebGoat/js/goatApp directory, specifically targeting the route handling files. The key file, GoatRouter.js, was found under goatApp/View. Here, I identified a suspicious route, 'test/:param': 'testRoute', which indicated a potential vulnerability. This route, left over from test code, was not secured for production. Combining this with the base route

```
start.mvc#lesson/
```

the compromised route was found out as

```
start.mvc#test/
```

. This highlighted the critical need for thorough security checks in web application development.

Your objective is to find the route and exploit it. First though, what is the base route? As an example, look at the URL `fc /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9`. The 'base route' in this case is: `start.mvc#lesson/` The `Cros` that are processed by the JavaScript route handler.

So, what is the route for the test code that stayed in the app during production? To answer this question, you have to c

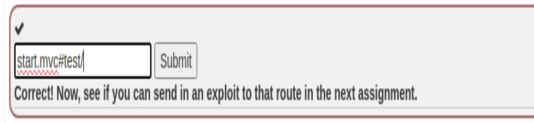


Fig. 4. Identifying potential for DOM-Based XSS

## VI. DOM-BASED XSS

For this task, first we need to find the specific route in WebGoat that reflected parameters without encoding, my objective was to trigger the `webgoat.customjs.phoneHome()` function. To achieve this, I crafted a URL that included JavaScript within the parameters, ensuring URL-encoding for proper execution. This URL was

```
http://localhost:8080/WebGoat/start.mvc#
test/<script>webgoat.customjs.phoneHome()
&%2Fscript>
```

Then navigating to this URL in a new tab and accessing the browser's console tab in Development Tools, the script executed successfully. This was confirmed by the appearance of a random number in the console, generated as a response from the `webgoat.customjs.phoneHome()` function, thereby executing a successful DOM-based XSS attack.

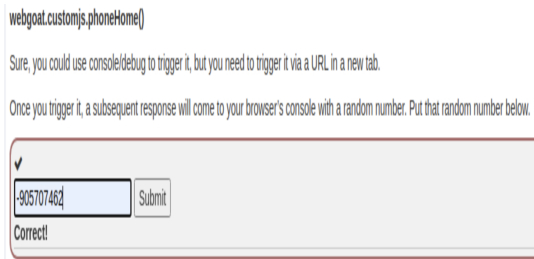


Fig. 5. DOM-Based XSS

For the next task, we were asked few questions related to Cross-site scripting (XSS), I completed the task successfully.

## VII. CONCLUSION

This lab provided valuable insights into the vulnerabilities related of Cross-site scripting. Cross-site scripting (XSS) enables attackers to run their own JavaScript in a victim's browser. It is a type of confused deputy attack, utilizing the web browser to execute critical tasks on a web application. Validating inputs on the server side is essential for security. XSS happens when user input, not properly validated, is included in an HTTP response. In a reflected XSS scenario, attackers create URLs containing the malicious script, which they distribute via other websites, emails, or other means to lure victims into clicking on them.

## REFERENCES

- [1] KVM  
<https://www.linux-kvm.org/page/MainPage..>
- [2] WebGoat  
[https://owasp.org/www-project-webgoat/..](https://owasp.org/www-project-webgoat/)
- [3] OverLeaf  
<https://www.overleaf.com/>.
- [4] Github  
<https://en.wikipedia.org/wiki/GitHub>.
- [5] RED  
<https://kb.iu.edu/d/apum>.
- [6] OWASP ZAP  
<https://www.zaproxy.org/>
- [7] Firefox DevTools  
<https://firefox-source-docs.mozilla.org/devtools-user/>