Offensive Security Using Burp Suite

José Torres
Master of Computer Science – IT Management and Information Security
Dr. Jeffrey Duffany
Department of Electrical and Computer Engineering
Polytechnic University of Puerto Rico

Abstract *— Burp Suite is a tool used by most professional web page pen tester. It has many features to help the pen tester do his job. The tool help students learn about different type of vulnerabilities like web cache poisoning, SQL injection, cross-site scripting (xss), and clickjacking attacks.*

Key Terms *— ethical hacking, Burp Suite, offensive security, web pen testing.*

INTRODUCTION

Burp Suite is a collection of tightly integrated tools that allow effective security testing of modern-day web applications [1]. Burp Suite is written in Java Language so basically the tool is cross-platform, Kali Linux already have Burp Suite installed on their OS image. There are two version of the tool: Community version which is free, and the professional version which cost $399 per year, per user. The main differences between the free and the pro version are:

- Burp Scanner
- The ability to save and restore your work
- Engagement tools, such as Target Analyzer, Content Discovery, and Task Scheduler

All other tools like intercepting traffic, brute forcing, etc., are available on the free version but work cannot be saved. However, the community version is great to start learning.

Burp Suite was used to test, evaluate its different feature, and the advantages on using Burp.

1. WEB CACHE POISONING USING BURP SUITE

Web cache poisoning is an advanced technique whereby an attacker exploits the behavior of a web server and cache so that a harmful HTTP response is served to other users. This technique involves two phases, the attacker must work out how to elicit a response from the back-end server that inadvertently contains dangerous payload. Once successful, the attacker must be sure that their response4 is cached and subsequently served to the intended victims [2].

Web ache sit between the user and the server, were they save and serve copies of certain responses (see Figure 1.1).

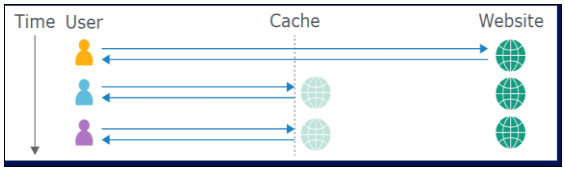

Figure 1.1 View of Three Users Fetching the Same Resource One After The Other [2].

1.2 Lab: Web Cache Poisoning With an Unkeyed Header

A lab from portswigger.com academy web poisoning section called “Lab: Web cache poisoning with an unkeyed header” was used.

Solution for completing the lab using [2]:

- With Burp running, load the website's home page.
- In Burp, go to "Proxy" > "HTTP history" and study the requests and responses that you generated. Find the GET request for the home page and send it to Burp Repeater. The result view is shown in Figure 1.2.

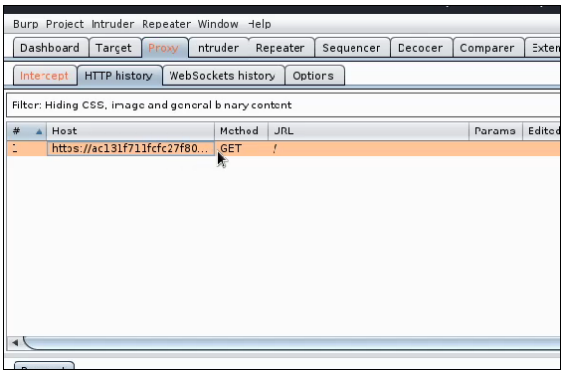

Figure 1.2
Get Request Page

- Add a cache-buster query parameter, such as "?cb=1234". See Figure 1.3.

Figure 1.3
Editing Cache-buster Query

- Add the X-Forwarded-Host header with an arbitrary hostname, such as "example.com", and send the request. See Figure 1.4.

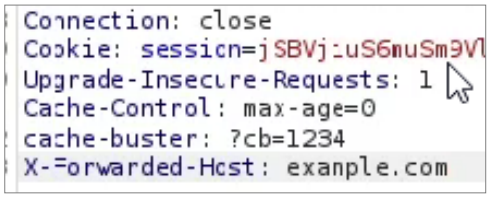

Figure 1.4
Adding "example.com" on X-Forwarded-Host

- Observe that the "X-Forwarded-Host" header has been used to dynamically generate an absolute URL for importing a JavaScript file stored at "/resources/js/tracking.js".
- Replay the request and observe that the response contains the header "X-Cache: hit". This tells us that the response came from the cache.
- Go to the exploit server and change the file name to match the path used by the vulnerable response: "/resources/js/tracking.js".
- In the body, enter the payload "alert(document.cookie)" and store the exploit.
- Open the "GET" request for the home page in Burp Repeater and remove the cache buster.
- Add the following header, remembering to enter your own exploit server ID: "X-Forwarded-Host: your-exploit-server-id.web-security-academy.net".
- Send your malicious request. Keep replaying the request until you see your exploit server URL being reflected in the response (see Figure 1.5) and "X-Cache: hit" in the headers.

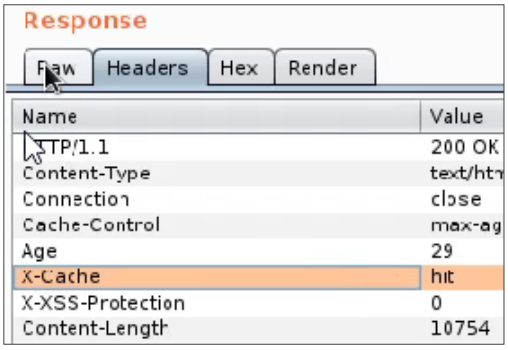

Figure 1.5
Server URL Response Page

- To simulate the victim, load the poisoned URL in your browser and make sure that the "alert()" is triggered. Note that you have to perform this test before the cache expires. The cache on this lab expires every 30 seconds.
- If the lab is still not solved, the victim did not access the page while the cache was poisoned. Keep sending the request every few seconds to re-poison the cache until the victim is affected and the lab is solved. Confirmation page is shown in Figure 1.6.

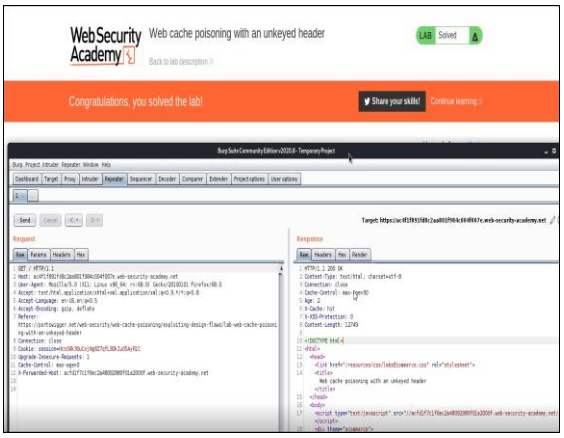

Figure 1.6
Confirmation page

1.3 Web Cache Poisoning With an Unkeyed Cookie

Another lab from the web poisoning section was used, but this time it is for unkeyed cookie.

Solution for completing the lab using [3]:

- With Burp running, load the website's home page.
- In Burp, go to "Proxy" > "HTTP history" and study the requests and responses that you generated. Notice that the first response you received sets the cookie "fehost=prod-cache-01" as shown in Figure 1.7.

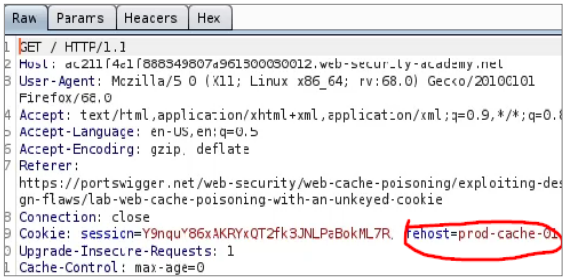

Figure 1.7
View of First Response

- Reload the home page and observe that the value from the "fehost" cookie is reflected inside a double-quoted JavaScript object in the response.
- Send this request to Burp Repeater and add a cache-buster query parameter.
- Change the value of the cookie to an arbitrary string and resend the request. Confirm that this string is reflected in the response.
- Place a suitable XSS payload in the fehost cookie, for example: "fehost=someString"-alert(1)- someString". See example n Figure 1.8.

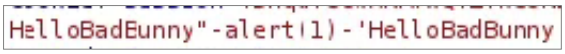

Figure 1.8
The String "HelloBadBunny" and the Payload

- Replay the request until you see the payload in the response and "X-Cache: hit" in the headers.
- Load the URL in your browser and confirm the "alert()" fires.
- Go back Burp Repeater, remove the cache buster, and replay the request to keep the cache poisoned until the victim visits the site and the lab is solved. Confirmation is viewed in Figure 1.9.

Figure 1.9
Confirmation of the Lab Completed

2. SQL Injection Using Burp Suite

SQL Injection it is a vulnerability that allow hackers to use queries of the application that makes to its database. The attacker can get the data that is not normally retrieve like data obliging to other users or access credential for the application. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure or perform a denial-of-service attack [4].

2.1 Impact of a Successful SQL Injection Attack

A successful attempt of sql injection can result in unauthorized access to data such as passwords

and other personal data like credit cards etc. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines [4].

2.2 Lab: SQL Injection Vulnerability in WHERE Clause Allowing Retrieval of Hidden Data

This lab contains a vulnerability that in any category of products can display interesting information from tables. To solve this lab, the following is needed using [4]:

- Use Burp Suite to intercept and modify the request that sets the product category filter.
- Modify the “category” parameter, giving it the value '+OR+1=1--. See Figure 2.1 for an example.

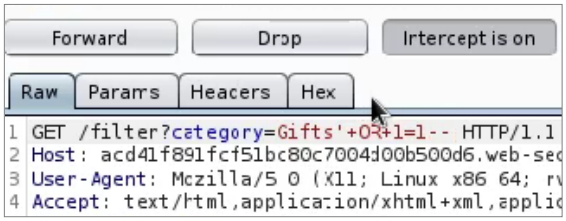

Figure 2.1
Example of a Sql Injection Query

- Submit the request and verify that the response now contains additional items. See Figure 2.2.

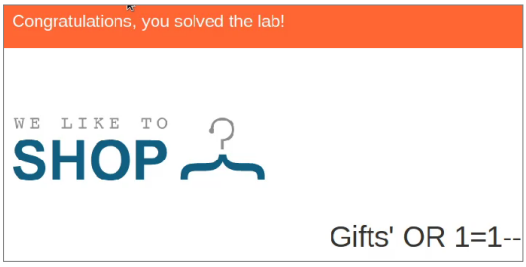

Figure 2.2
Confirmation Page of the Lab Completed

2.3 Lab: SQL Injection UNION Attack, Retrieving Data From Other Tables

This lab contains an SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so a UNION attack to retrieve data from other tables can be used. To solve the lab, follow these instructions [4]:

- Use Burp Suite to intercept and modify the request that sets the product category filter.
- Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter (see Figure 2.2): '+UNION+SELECT+'abc','def'--.

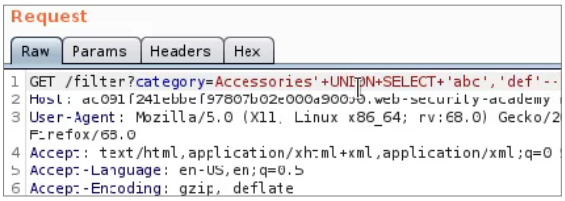

Figure 2.2
Screenshot of the Parameter

- Verify that the application's response contains usernames and passwords. See Figure 2.3 for confirmation page view.

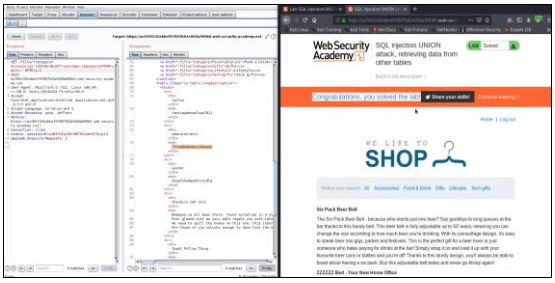

Figure 2.3
Confirmation Page of this lab

3. Cross-site Scripting (XSS) Using Burp Suite

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user can perform, and to access any of the user's data. If the victim user has privileged access within the application, then

the attacker might be able to gain full control over all the application's functionality and data [5].

3.1 How Does XSS Work?

XSS works, basically, by manipulating a web site that returns a malicious JavaScript to users and the code get executed in the victim's browser so that the attacker compromises their interaction with the application.

There are three types of XSS attacks:

- Reflected XSS
- Stored XSS
- DOM-based XSS

Reflected XSS is the simplest variety of cross-site scripting. It arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way [5].

Stored XSS (also known as persistent or second order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way. The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources; for example, a webmail application displaying messages received over SMTP, a marketing application displaying social media posts, or a network monitoring application displaying packet data from network traffic [5].

DOM-based XSS (also known as DOM XSS) arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM.

3.2 Lab: Reflected XSS Into HTML Context With Nothing Encoded

This lab contains a simple reflected cross-site scripting vulnerability in the search functionality.

To solve this lab [5]:

- Copy and paste the following into the search box: <script>alert(1)</script> as shown in Figure 3.1.

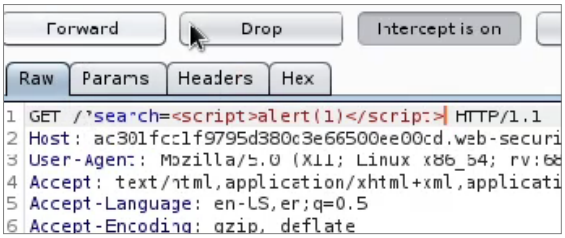

Figure 3.1
Writing on the Search Parameter the Javascript Alert Box

- Click "Search". The screen will show confirmation as in Figure 3.2.

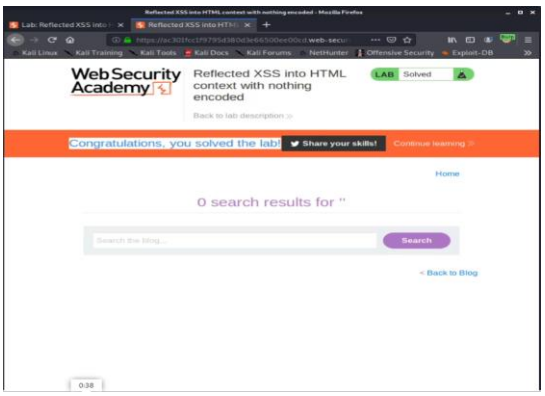

Figure 3.2
Lab Confirmation Screen

3.3 Lab: Reflected XSS Into a JavaScript String With Single Quote and Backslash Escaped

This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality. The reflection occurs inside a JavaScript string with single quotes and backslashes escaped.

To solve this lab using [5]:

- Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
- Observe that the random string has been reflected inside a JavaScript string.
- Try sending the payload test'payload and observe that the single quote gets backslash-escaped, preventing from breaking out of the string.

- Replace your input with the following payload to break out of the script block and inject a new script: </script><script>alert(1)</script>. See Figure 3.3.

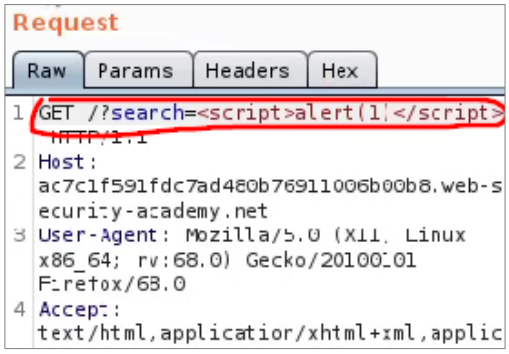

Figure 3.3
Injecting Script

- Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in your browser. When you load the page it should trigger an alert. See Figure 3.4.

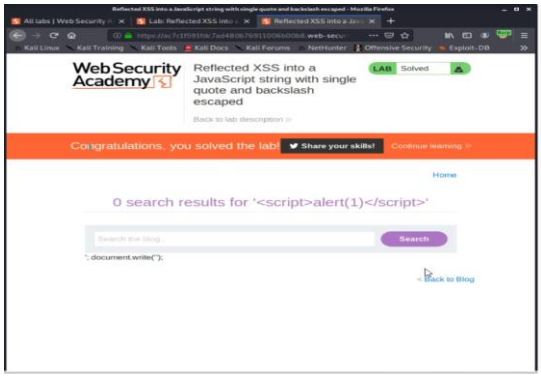

Figure 3.4
Lab Confirmation Screen

4. CLICKJACKING USING BURP SUITE

Clickjacking is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy website. An example for clickjacking would be that A web user accesses a decoy website (for example a link provided by an email) and clicks on a button to win a prize. Unknowingly, they have been deceived by an attacker into pressing an alternative hidden button and this results in the payment of an account on another site. This is an example of a clickjacking attack. The technique depends upon the incorporation of an invisible, actionable web page (or multiple pages) containing a button or hidden link, that could be within an iframe. The iframe is overlaid on top of the user's anticipated decoy web page content. This attack differs from a CSRF attack in that the user is required to perform an action such as a button click whereas a CSRF attack depends upon forging an entire request without the user's knowledge or input [6].

4.1 Lab: Basic Clickjacking With CSRF Token Protection

This lab contains login functionality and a delete account button that is protected by a CSRF token. A user will be clicking on "click" on a decoy website and the goal of the lab is to entice the user into deleting their account [6].

To solve this lab using [6]:

- Login to the account on the target website. Login page is in Figure 4.1.

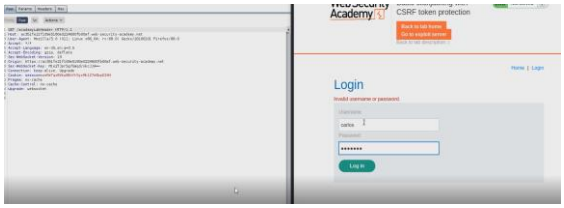

Figure 4.1
Loging in on the Target Website

- Use the following HTML template and provide the details as follows:
  - Replace $url with the URL for the target website account page in the iframe.
  - Substitute suitable values in pixels for the $height_value and $width_value variables of the iframe (we suggest 700px and 500px respectively).
  - Substitute suitable values in pixels for the $top_value and $side_value variables of the decoy web content so that the "delete account" button and the "click me" decoy action align (we suggest 300px and 60px respectively).
  - Set the opacity value $opacity to ensure that the target iframe is transparent.

Initially, use an opacity of 0.1 so that you can align the iframe actions and adjust the position values as necessary. For the submitted attack a value of 0.0001 will work. See Figure 4.2.

```
<style>
    iframe {
        position:relative;
        width:700px;
        height: 500px;
        opacity: 0.0001;
        z-index: 2;
    }
    div {
        position:absolute;
        top:300px;
        left:60px;
        z-index: 1;
    }
</style>
<div>Test me</div>
<iframe src="$url"></iframe>
```

Figure 4.2
CSS Template for the Payload

- Go to the exploit server, paste your exploit HTML into the "Body text" box, and click "Store".
- Click "View stored response".
- Hover over "Test me" and ensure the cursor changes to a hand indicating that the div element is positioned correctly. If not, adjust the position of the div element by modifying the top and left properties of the style sheet. Result is shown in Figure 4.3.

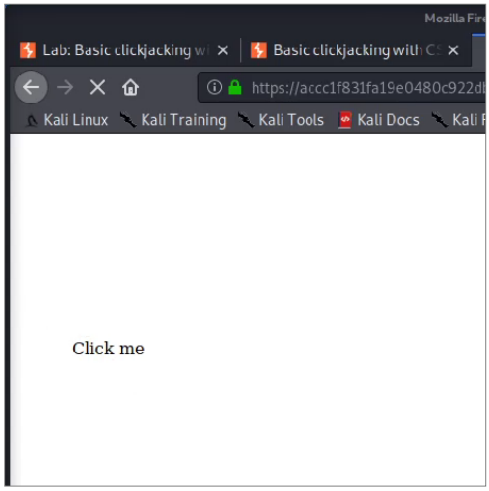

Figure 4.3
View of Display

- Once you have the div element lined up correctly, change "Test me" to "Click me" and click "Store".
- Now click on "deliver exploit to victim" and the lab should be solved. Lab confirmation is shown in Figure 4.4.

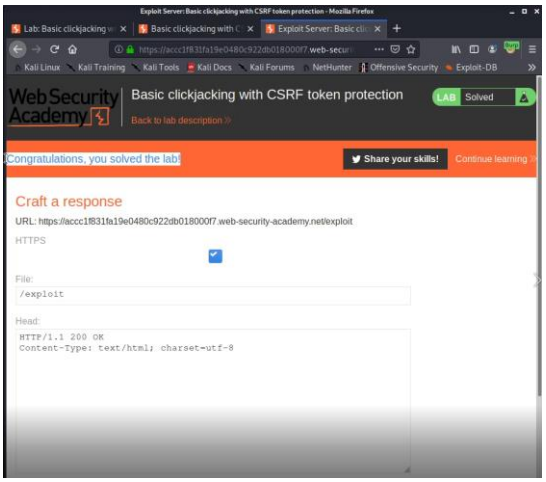

Figure 4.4
Lab Confirmation.

4.2 Lab: Clickjacking with form input data prefilled from a URL parameter

This lab extends the basic clickjacking example in Lab: Basic clickjacking with CSRF token protection. The goal of the lab is to change the email address of the user by prepopulating a form using a URL parameter and enticing the user to click on a "update email" button without the user's knowledge.

To solve this lab using [6]:

- Login to the account on the target website.
- Use the following HTML template and provide the details as follows:
    - Replace $url with the URL for the target website change email page in the iframe.
    - Substitute suitable values in pixels for the $height_value and $width_value variables of the iframe (we suggest 700px and 500px respectively).
    - Substitute suitable values in pixels for the $top_value and $side_value variables of the decoy web content so that the "update

email" button and the "Test me" decoy action align (we suggest 400px and 80px respectively).

- Set the opacity value $opacity to ensure that the target iframe is transparent. Initially, use an opacity of 0.1 so that you can align the iframe actions and adjust the position values, as necessary. For the submitted attack a value of 0.0001 will work. Figure 4.5 shows CSS template.

```
<style>
  iframe {
      position:relative;
      width:$width_value;
      height: $height_value;
      opacity: $opacity;
      z-index: 2;
  }
  div {
      position:absolute;
      top:$top_value;
      left:$side_value;
      z-index: 1;
  }
</style>
<div>Test me</div>
<iframe src="$url?email=hacker@attacker-website.com"></if
```

Figure 4.5
CSS Template

- Go to the exploit server, paste your exploit HTML into the "Body text" box, and click "Store".
- Click "View exploit". Result is shown in Figure 4.6.

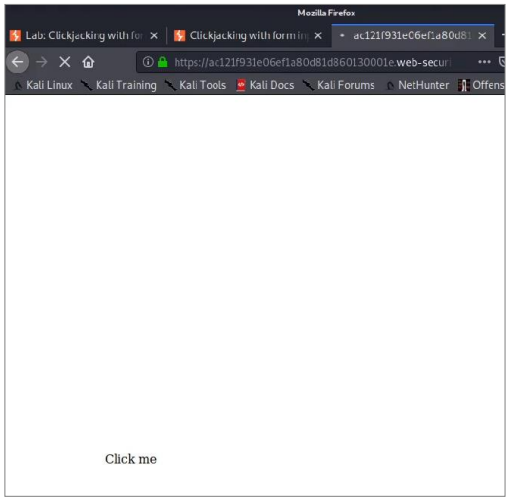

Figure 4.6
View Exploit

- Hover over "Test me" and ensure the cursor changes to a hand indicating that the div element is positioned correctly. If not, adjust the position of the div element by modifying the top and left properties of the style sheet.
- Once you have the div element lined up correctly, change "Test me" to "Click me" and click "Store".
- Now click on "deliver exploit to victim" and the lab should be solved. Lab confirmation page is shown n Figure 4.7.

Figure 4.7
Lab Confirmation Page

5. ACCESS CONTROL VULNERABILITIES AND PRIVILEGE ESCALATION USING BURPSUITE

Access control (or authorization) is the application of constraints on who (or what) can perform attempted actions or access resources that they have requested. In the context of web applications, access control is dependent on authentication and session management [7]:

- Authentication identifies the user and confirms that they are who they say they are.
- Session management identifies which subsequent HTTP requests are being made by that same user.
- Access control determines whether the user is allowed to carry out the action that they are attempting to perform.

Broken access controls are a commonly encountered and often critical security

vulnerability. Access control design decisions have to be made by humans and the potential for errors is high.

Some types of access control vulnerabilities Definitions:

Vertical access controls: Mechanisms that restrict access functionality that is not available to other users.

Horizontal access controls: Mechanism that restrict access to resources to the users that can access those resources.

Context-dependent access controls: Restrict access to functionality and resources based upon the state of the application [7].

5.1 Lab: User Role Controlled by Request Parameter

This lab has an admin panel at /admin, which identifies administrators using a forgeable cookie.

To solve this lab:

- Browse to /admin and observe that you cannot access the admin panel (see Figure 5.1).

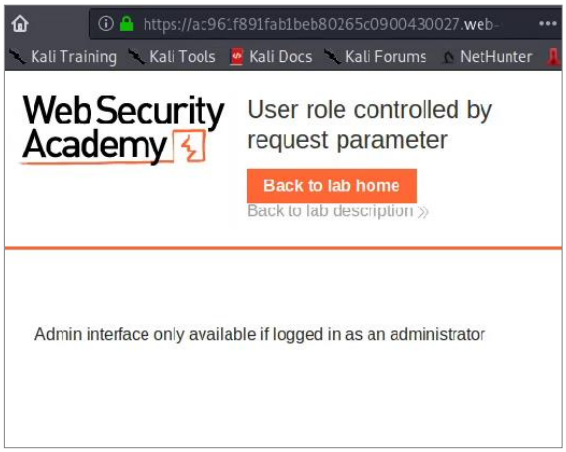

Figure 5.1
Cannot Access Admin Panel

- Browse to the login page.
- In Burp Proxy, turn interception on and enable response interception.
- Complete and submit the login page and forward the resulting request in Burp.
- Observe (see Figure 5.2) that the response sets the cookie Admin=false. Change it to Admin=true.

Figure 5.2
Changing Admin to True

- Load the admin panel and delete “carlos”.

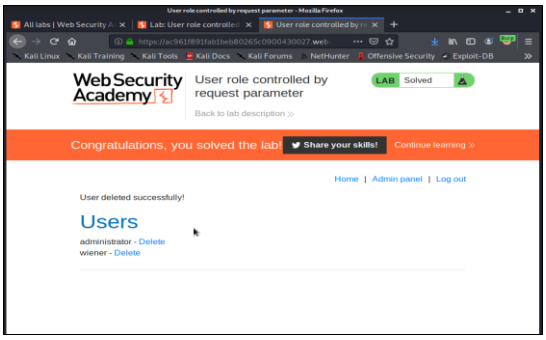

Figure 5.3
Lab Comfirmation

SUMMARY AND CONCLUSION

Burp Suite, a collection of integrated tools used by most of web pen tester is a great tool to start pen testing. Their website portswigger is great place to start learning and practicing legally like web cache poisoning, sql injection, cross-site scripting (XSS), clickjacking, and access control vulnerabilities and privilege escalation attacks. As you can see on this articles Burp Suite mainly is used for intercepting a website in order to analyze its traffic, but it can be used to scan websites (pro version only), used payloads like brute forcing (community version works but the pro version is a lot faster), etc. Because there are too many labs, some were not covered, but the most important were selected and common vulnerabilities using Burp Suite could be demonstrated. The labs chosen for this investigation have the necessary information to start on Burp Suite and ethical hacking.

REFERENCES

[1] A. Mahajan, *Burp Suite Essentials*, Birmingham, United Kingdom: Packt Publishing 2014.

[2] Port Swigger Academy, "Web Cache Poisoning Lab 1: Web cache poisoning with an unkeyed header," P*ortSwigger.net*, n.d. [Online]. Available: https://portswigger.net/web-security/web-cache-poisoning/exploiting-design-flaws/lab-web-cache-poisoning-with-an-unkeyed-header.

[3] Port Swigger Academy, "Web Cache Poisoning Lab 2: Web cache poisoning with an unkeyed cookie," *PortSwigger.net*, n.d. [Online]. Available: https://portswigger.net/web-security/web-cache-poisoning/exploiting-design-flaws/lab-web-cache-poisoning-with-an-unkeyed-cookie.

[4] Port Swigger Academy, "SQL Injection." *PortSwigger.net*, n.d. [Online]. Available: https://portswigger.net/web-security/sql-injection.

[5] Port Swigger Academy, "Cross-site scripting." *PortSwigger.net*, n.d. [Online]. Available: https://portswigger.net/web-security/cross-site-scripting.

[6] Port Swigger Academy, "Clickjacking (UI redressing)." *PortSwigger.net*, n.d. [Online]. Available: https://portswigger.net/web-security/clickjacking.

[7] Port Swigger Academy, "Access control vulnerabilities and privilege escalation." *PortSwigger.net*, n.d. [Online]. Available: https://portswigger.net/web-security/access-control.