CEH v12 Lesson 6 : Web Application Exploitation Concepts

Learning Outcomes

In this module, you will complete the following exercises:

- Exercise 1 — Web Application Concepts
- Exercise 2 — Web Application Threats
- Exercise 3 — Web Application Hacking Methodology
- Exercise 4 — Footprint and Analyze web Infrastructure

After completing this module, you will be able to:

- Sniff Passwords
- Use Medusa to Crack Passwords
- Perform Broken Authentication Attacks
- Conduct OS Command Injection Attack
- Perform Server-side Includes Injection Attack (SSI)
- Perform Cross-site Scripting Attack
- Perform Cross-site Scripting (XSS) — Reflected (HREF) Attack
- Enumerate web Applications Using Wafwoof
- Perform Website Enumeration using Nmap
- Detect the IP addresses Using the Host Command
- Perform Information Gathering Using Legion

After completing this module, you will have further knowledge of:

- Web Applications and Their Advantages
- Web Application Architecture

- Web Services
- Vulnerability Stack in a web Application
- OWASP Top 10
- Web Application Hacking Methodology Concepts

Lab Duration

It will take approximately **1 hour and 30 minutes** to complete this lab.

Exercise 1 — Web Application Concepts

Most organizations offer their services in the form of a web application and can encompass many different services, such as a client-facing application or a collaboration platform that the users can use to communicate on a project's status. In most cases, access to these web applications is available through subscription.

In this exercise, you will learn about web application concepts.

Learning Outcomes

After completing this exercise, you will have further knowledge of:

- Web Applications and Their Advantages
- Web Application Architecture
- Web Services

Web Applications and Their Advantages

Most users may think of a web application as a website, but an application and website are different. Historically, a website usually consists of static web pages that need to be updated by a web developer when content changes are necessary. A web application is also on the web server, but consists of dynamic web pages that are created by server-side code such as PHP, Ruby, or other languages. Content is driven by user requests.

This interaction has multiple steps. First, the server-side code processes the user request. It then creates a SQL statement that is sent to the back-end database requesting the information the user wants. The database responds by sending only the requested data back to the web application. The web application processes the response and generates HTML code in a user-friendly format. This is then sent to the user's browser

for display and consumption. The interaction between the users and the web application allows each user to customize the content they consume, search, modify their profiles, place orders, and communicate with other users. Let's look at some of the key advantages of web applications.

- They are accessible through the Internet. The users can access them anytime from anywhere if they have an active subscription.
- Web applications can be accessed from different devices. For example, a user can access them from a desktop, laptop, or mobile.
- Most web applications can be customized based on user needs, and there is no dependency on developers. A user can customize the look and feel based on their specific requirements.
- Most web applications are equipped with reporting functions, allowing users to view statistics. For example, a user may want to view a report on the money spent on purchasing certain items through an online banking application.

Web Application Architecture

A web application consists of three layers, which are:

- **Presentation Layer**: also known as a client layer (or front-end), which represents information to a user. This layer includes systems used to fetch required information from the application. When a user requests something while using the application, the browser sends a request to the web server, which will respond to the request.
- **Business Logic Layer**: defines the flow of data within an application. Data is stored in a database and must be retrieved and saved. These actions are driven by user actions and are performed based on the business logic that the developers build into the application.
- **Database Layer**: contains the database and database server that hold the user's data. This data is saved or retrieved from the database based on the user's actions performed within the application.

Web Services

A web service allows communication between two applications, even if they are developed in different languages. For example, your website or application may want to

obtain data from a data provider such as a weather station. Since the data exists on another organization's server, you mostly likely don't have direct access to that data. That organization can make that data available to users by making a web service available. This access may be free or require a subscription. Either way, you have access using some rather simple coding techniques.

This is because a web service uses one of two key messaging protocols used with services, Simple Object Access Protocol (SOAP) and Representational State Transfer (REST). SOAP is an older protocol. It is based on XML and is highly structured. REST is now heavily used and works with various HTTP 1.1 verbs, such as GET, POST, PUT, and DELETE.

A web service is made up of three components, which are:

- **Service Provider**: the platform that hosts web services.
- **Service Requester**: the client that needs to use the web service.
- **Service Registry**: the location that the service provider uses to load the service descriptions, including the web service interface and its implementation details. It includes the details of network locations, bindings, and data types.

Exercise 2 — Web Application Threats

web application threats are common. They can occur for various reasons, and if not handled properly, they lead to various unfavorable outcomes, such as loss of data or application downtime. These vulnerabilities can exist because of various reasons, such as:

- The web application environment, such as the server, is vulnerable to a web application exploitation.
- The web application communicates with a vulnerable application, which leads to exploitation.
- There are inherent vulnerabilities within the web application that can be exploited.
- There are misconfigurations or programming flaws that lead to web application exploitation.

In this exercise, you will learn about various web application vulnerabilities.

Learning Outcomes

After completing this exercise, you will be able to:

- Sniffing the Passwords
- Use Medusa to Crack Passwords
- Perform Broken Authentication Attacks
- Conduct OS Command Injection Attack
- Perform Server-side Includes Injection Attack (SSI)
- Perform Cross-site Scripting Attack
- Perform Cross-site Scripting (XSS) — Reflected (HREF) Attack

After completing this exercise, you will have further knowledge of:

- Vulnerability Stack in a web Application
- OWASP Top 10

Vulnerability Stack in a Web Application

The web application infrastructure consists of seven different layers. Each layer can contain vulnerabilities that allow an attacker to exploit an application:

Layer 7 — Web ApplicationLayer 6 — Third-party ComponentsLayer 5 — Web ServerLayer 4 — DatabaseLayer 3 — Operating SystemLayer 2 — NetworkLayer 1 — Security

Figure 2.1: Diagram showing the 7 layers of infrastructure; Web Aapplications, Third-Party Components, Web Servers, Databases, Operating Systems, Networks, and Security.

Let’s look at each layer in detail:

Layer 7 — Web Application Vulnerabilities at this layer are due to flaws in business logic. There could also be coding flaws that can lead to attacks, such as SQL Injection or cross-site scripting.**Layer 6 — Third-party Components**Vulnerabilities exist due to integration of the third-party components. A web application may not have a technical flaw or vulnerability, but can still be exploited due to vulnerable components which includes things such as an add-on or web service provided by a third party.**Layer 5 —**

Web Server Web servers are prone to footprinting and reconnaissance, allowing an attacker to gain a lot of information. For example, an attacker finds the name and version of a web server and can search for vulnerabilities in the vulnerabilities database.**Layer 4 — Database**A database, if not patched, can have vulnerabilities that tools such as SQLMAP can exploit.**Layer 3 — Operating System**An operating system must be patched when updates are available. An attacker can use various methods to attack an OS (such as social engineering), to deliver malware which will exploit an operating system or its vulnerabilities.**Layer 2 — Network**Attackers can attack edge routers and firewalls with DoS attacks. If not configured properly, switches can be flooded with requests to overflow their CAM tables. Attackers also attempt to sniff network traffic./td>**Layer 1 — Security**A network can have an intrusion detection system (IDS) or intrusion prevention system (IPS) to detect and block malicious traffic. However, attackers often use IDS/IPS and firewall evasion methods to generate no alert.

OWASP Top 10

In many scenarios, organizations use off-the-shelf applications. In other scenarios, the organizations use a mix of off-the-shelf and custom applications. The patches and updates for the off-the-shelf applications are created and released by the vendor that created the application. It is now the organization's responsibility to update their existing off-the-shelf applications to patch these applications. In the case of custom applications, this is not the scenario. The custom applications are developed to meet a specific business need. An organization may have an in-house development team to create the application or outsource it to another vendor. Releasing updates is a common issue with custom applications. The vendors do not provide updates in most cases. Therefore, these inherent vulnerabilities will continue to be present until the application is used.

There are several known web application vulnerabilities. Open web Application Security Project, more commonly known as OWASP, releases the top 10 web application vulnerabilities, which are released after every few years. This data is collected from various organizations through extensive research, and then the top 10 web applications are selected. Remember — there are hundreds of web application vulnerabilities, and therefore, when doing a penetration test, you can focus on the key ones but do not ignore to test for the other vulnerabilities. The top 10 web application vulnerabilities of 2021 released by OWASP are:

- **A01:202**: Broken Access Control

- **A02:202**: Cryptographic Failures
- **A03:2021**: Injection
- **A04:2021**: Insecure Design
- **A05:2021**: Security Misconfiguration
- **A06:2021**: Vulnerable and Outdated Components
- **A07:2021**: Identification and Authentication Failures
- **A08:2021**: Software and Data Integrity Failures
- **A09:2021**: Security Logging and Monitoring Failures
- **A10:2021**: Server-Side Request Forgery

Task 1 — Sniffing Passwords

Even though sniffing the password does not fall into “offline password cracking,” it is a great method to capture the password transmitted in unencrypted form.

In this task, you will learn about sniffing passwords. To do this, perform the following steps:

Step 1

Ensure you have powered on all the devices listed in the introduction and connect to **PLABKALI01**.

Log in using the following credentials:

Username:

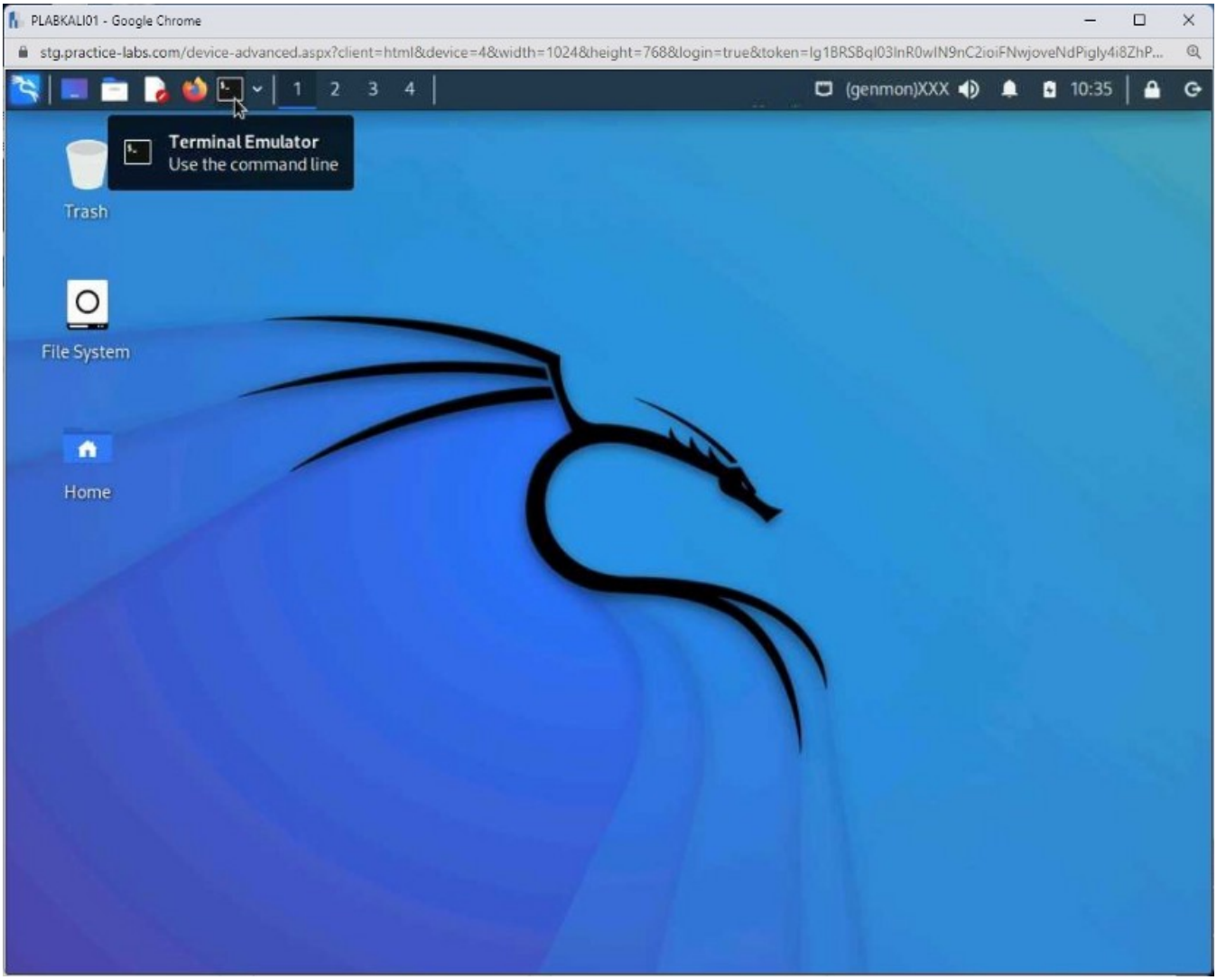
```
root
```

Password:

Password

The desktop of **PLABKALI01** is displayed.

Open a new terminal window by clicking the **Terminal Emulator** icon on the taskbar.



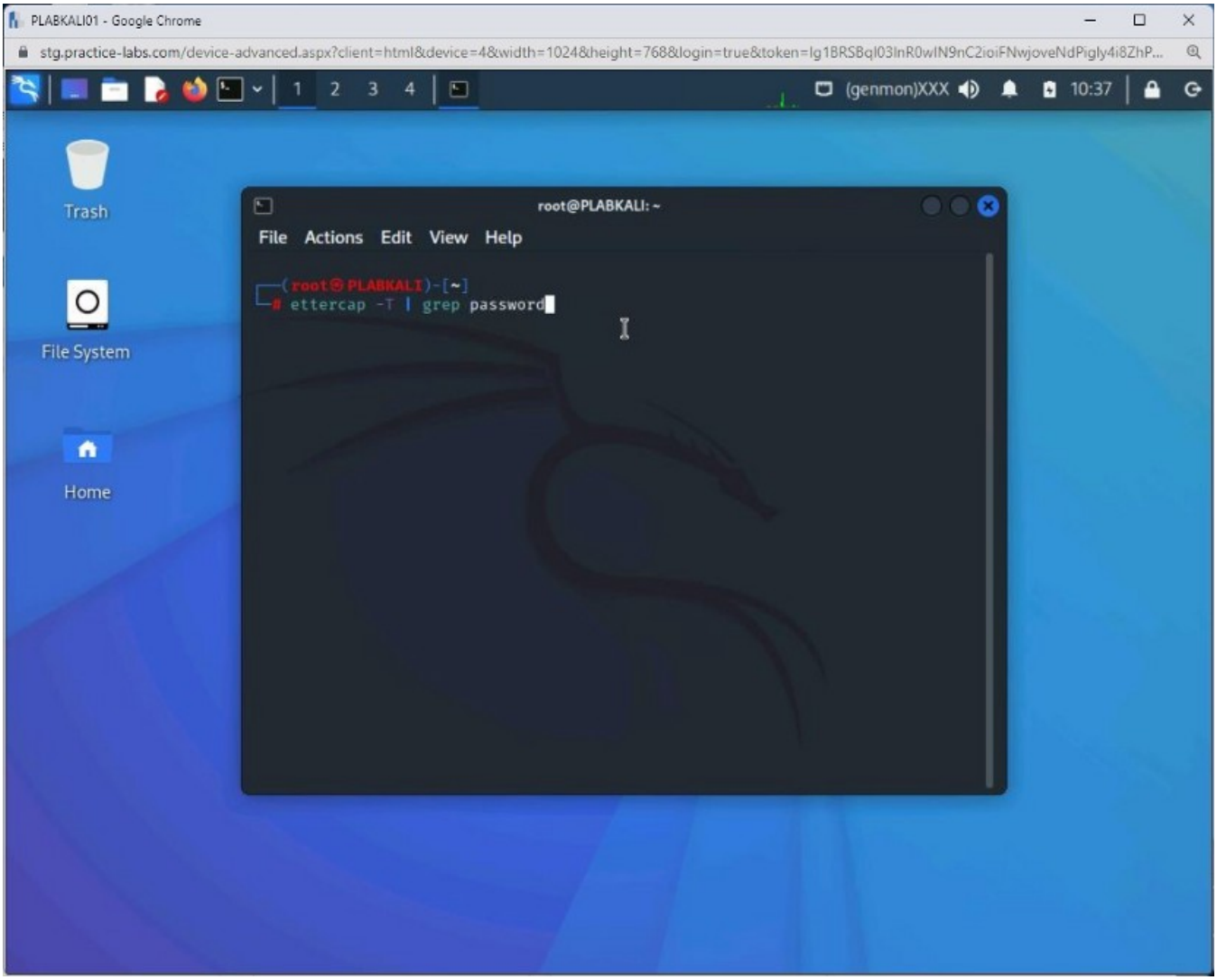
Step 2

The terminal window is displayed.

The **Ettercap** tool is used to capture a password from an unencrypted session. Type the following command:

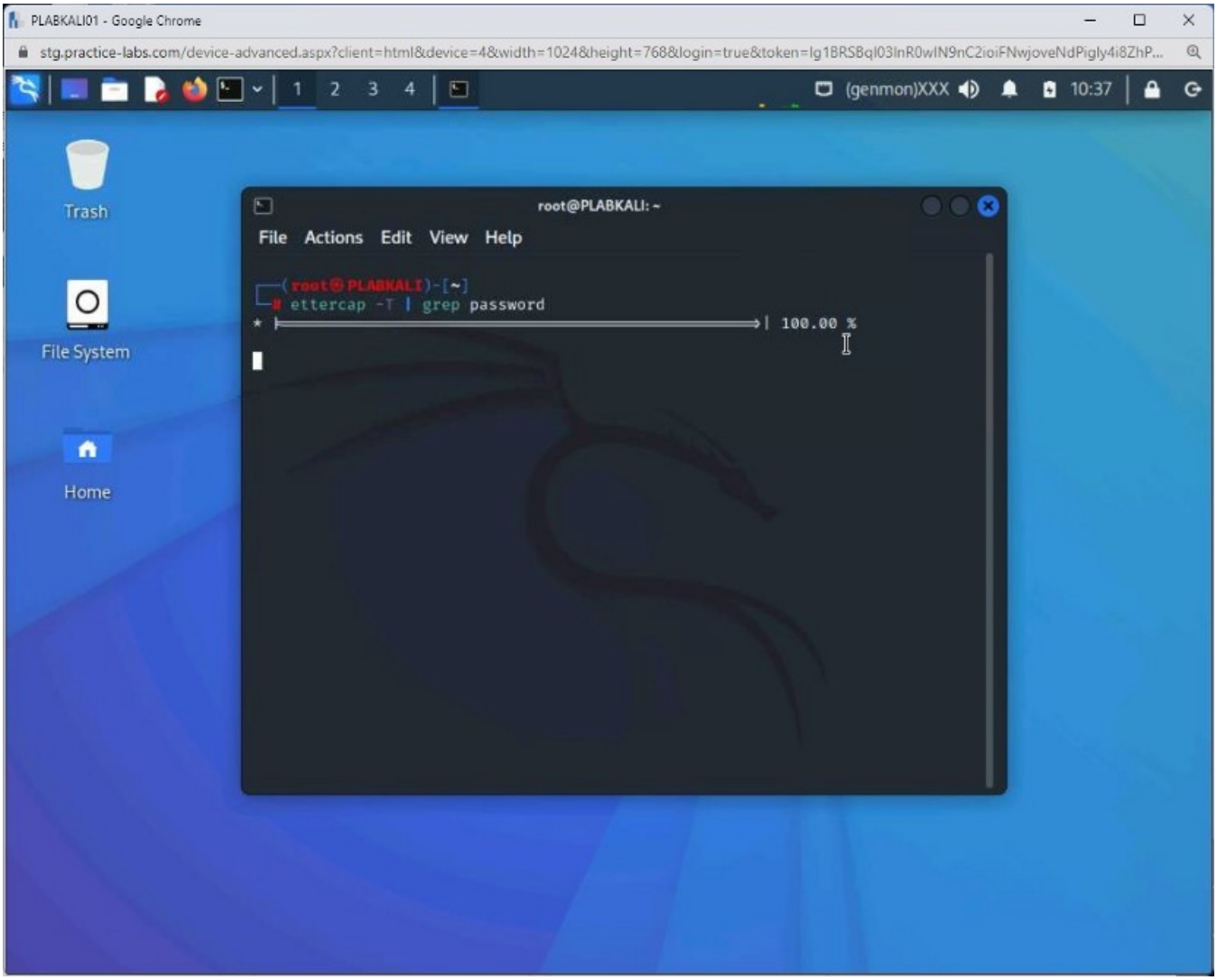
```
ettercap -T | grep password
```

Press **Enter**.



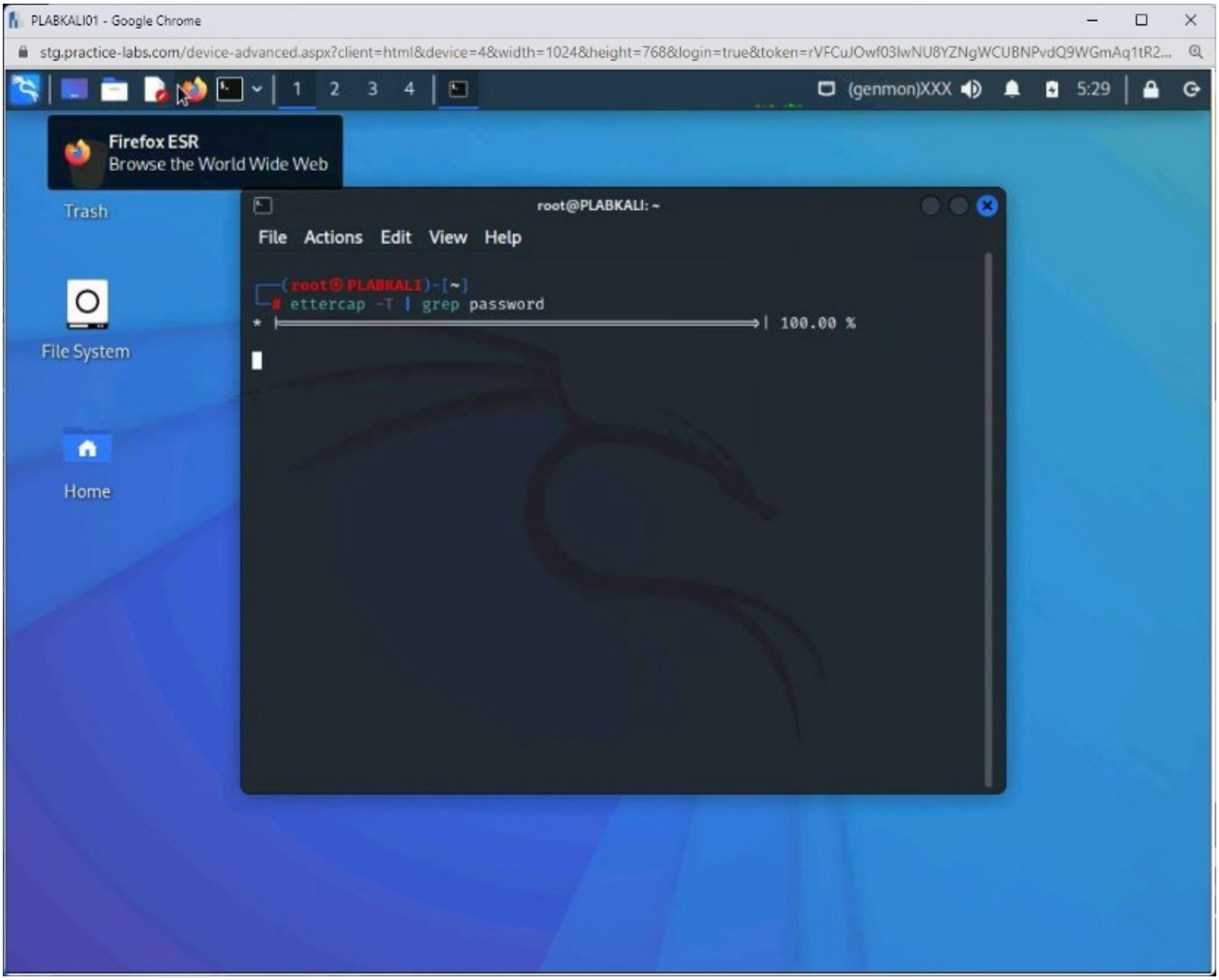
Step 3

The sniffing process starts.



Step 4

Click the **Firefox ESR** icon on the taskbar.

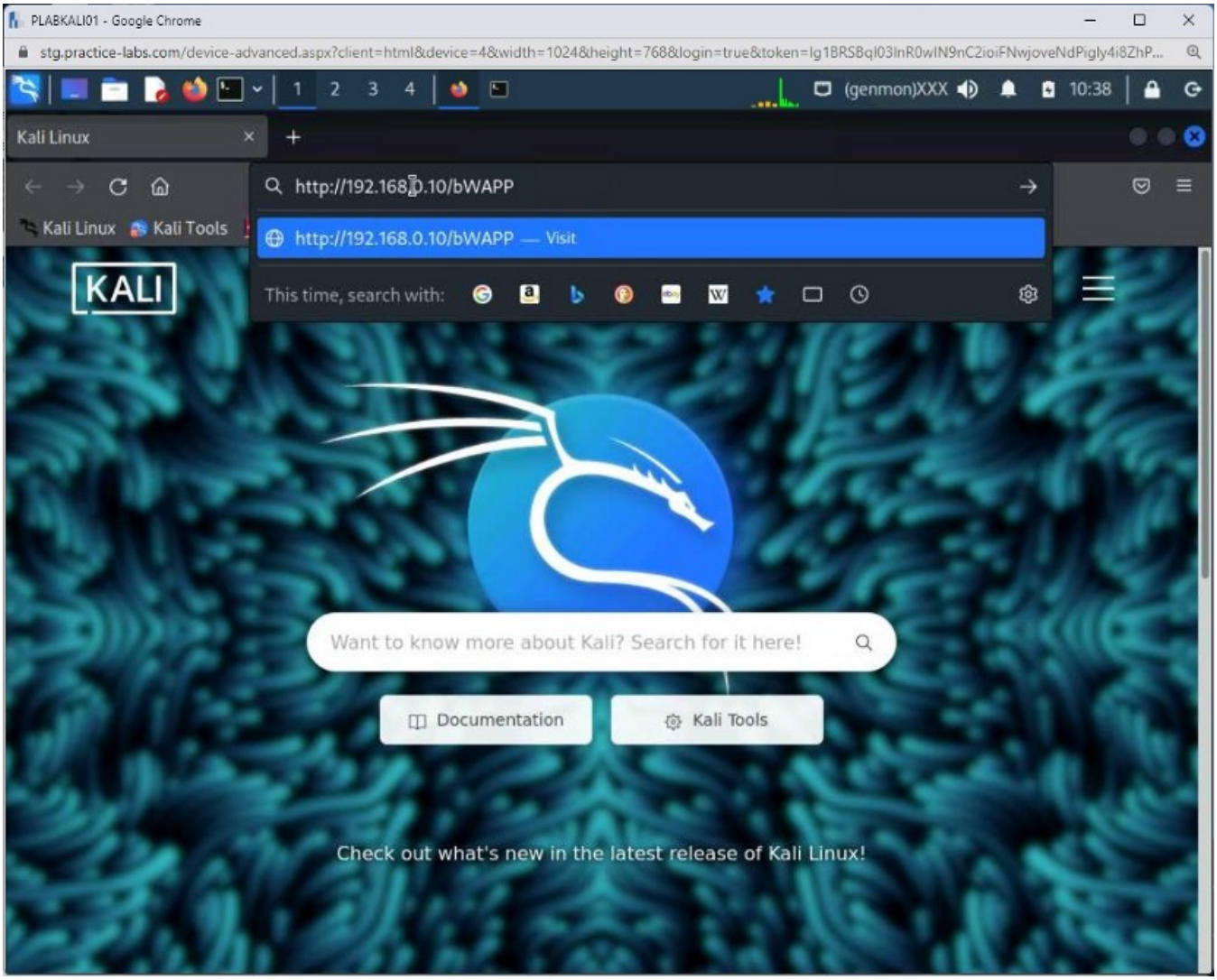


Step 5

The Firefox window is displayed. In the address bar, type the following URL:

`http://192.168.0.10/bWAPP`

Press **Enter**.



Step 6

The login page is displayed.

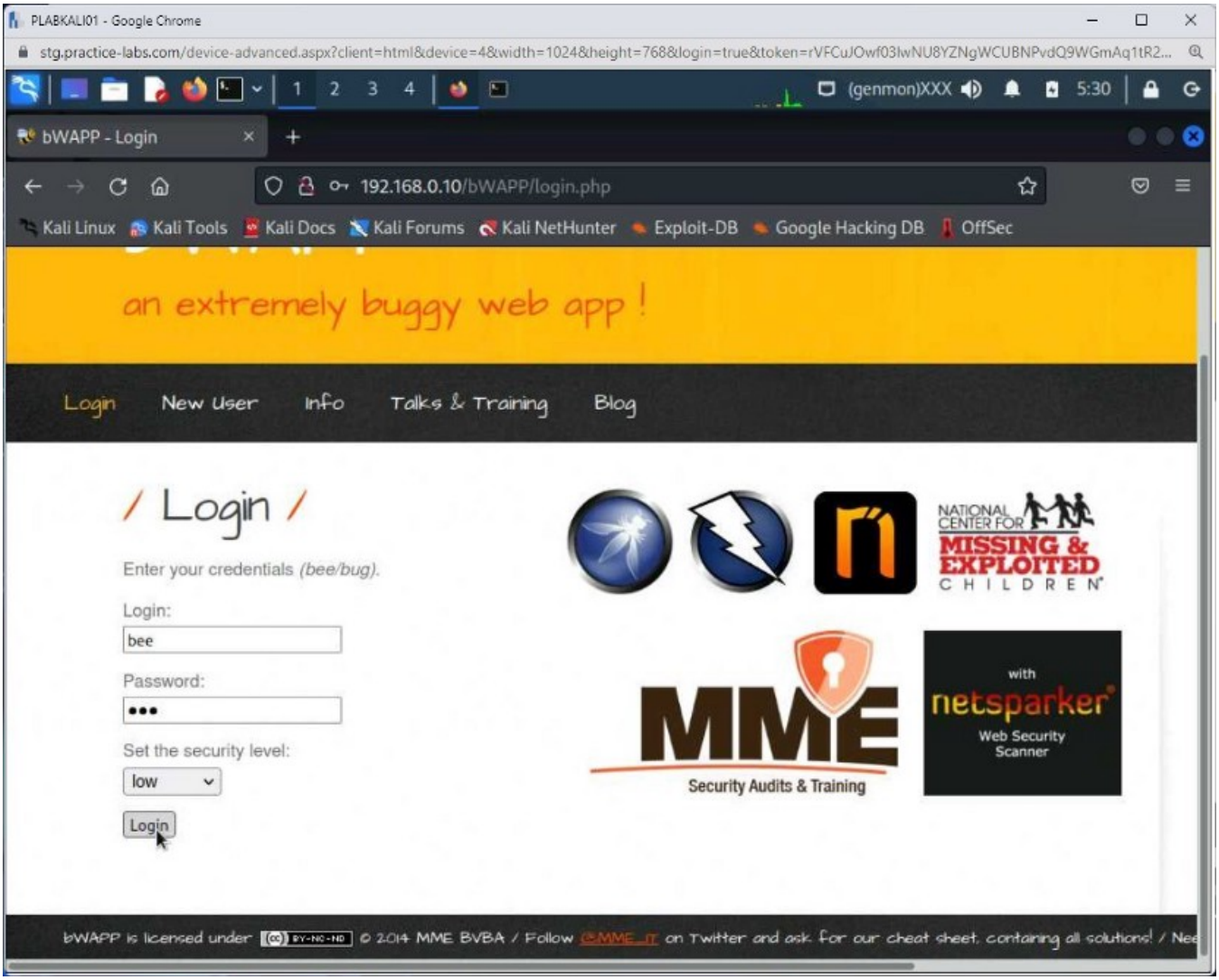
In the **Username** text box, type the following:

```
bee
```

In the **Password** text box, type the following:

bug

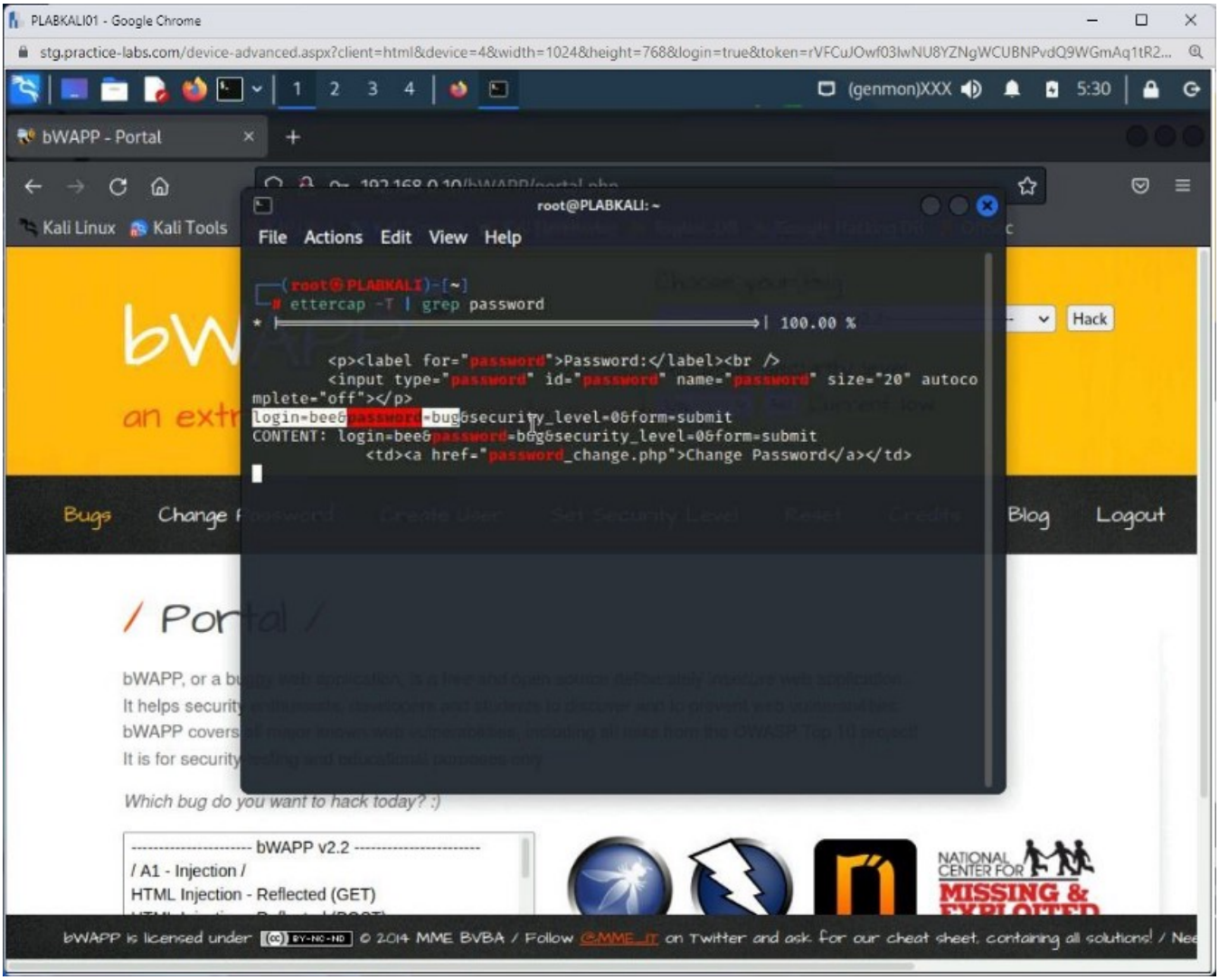
Click **Login**.



Step 7

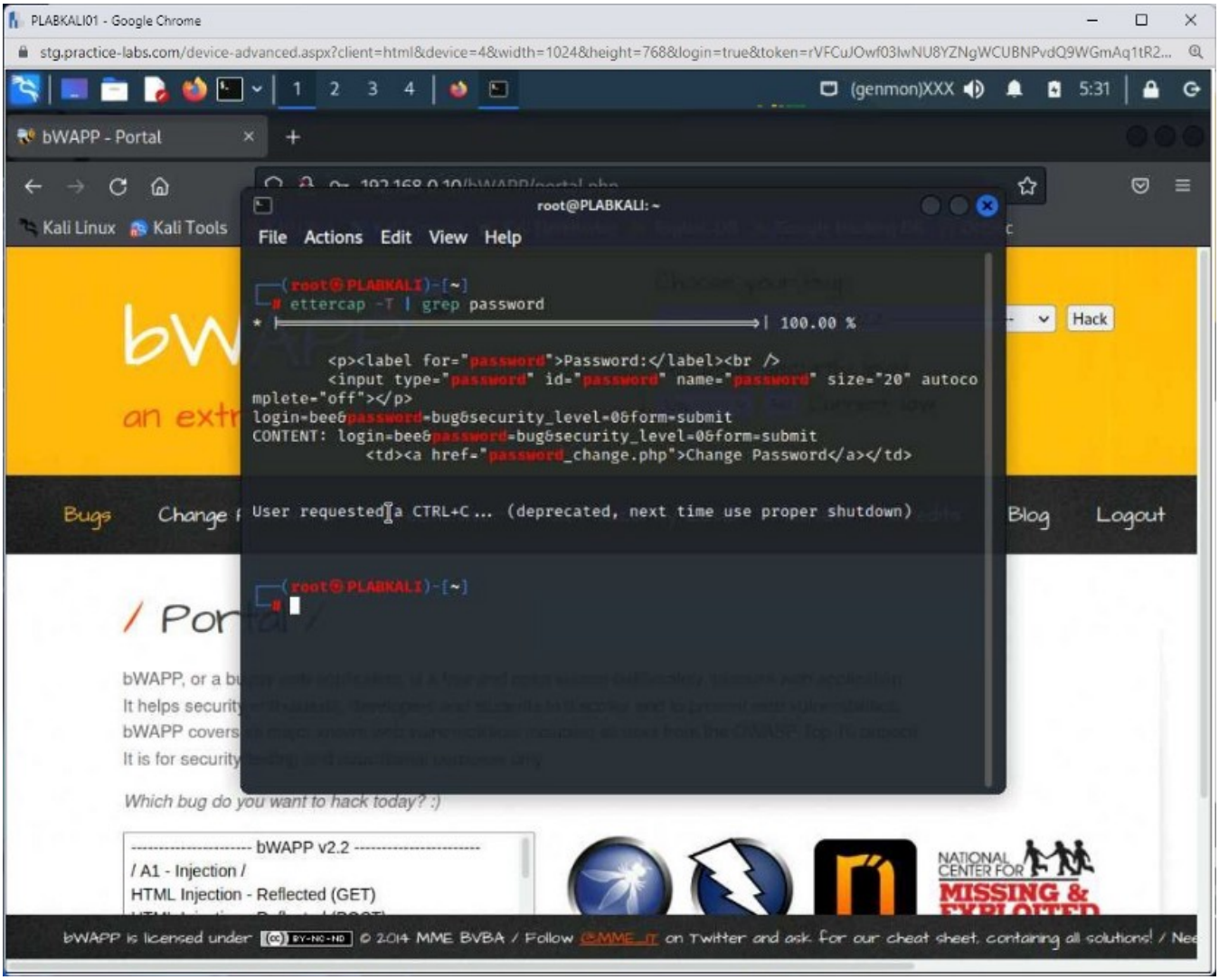
Click the terminal window to bring it to the foreground.

Notice that the **username** and **password** are now captured.



Step 8

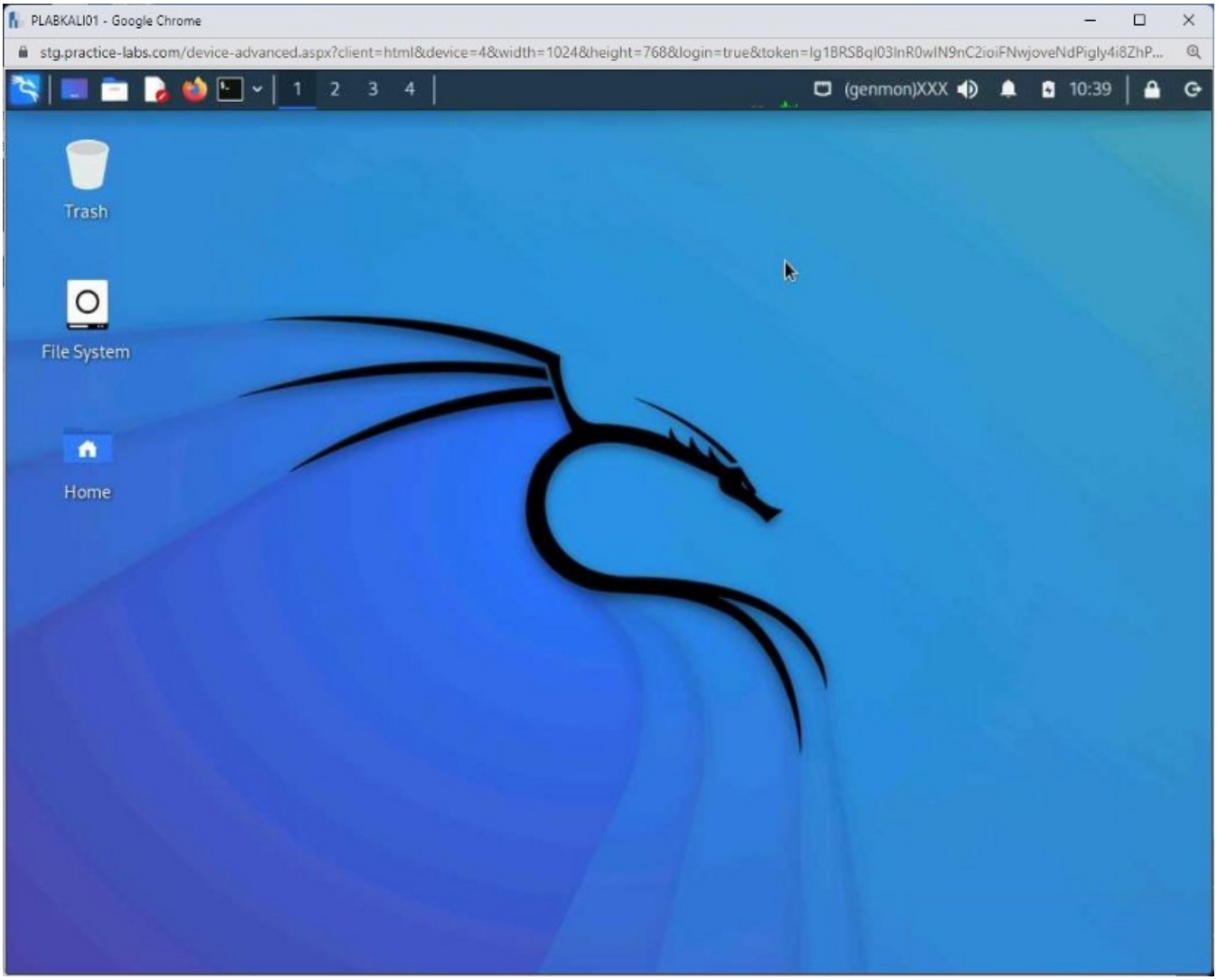
Press the **Ctrl + C** keys to break the sniffing process.



Step 9

Close all open windows.

You are now back on the **PLABKALI01** desktop.



Close the Firefox ESR window. Keep the terminal window open.

Task 2 — Use Medusa to Crack Passwords

Medusa is a login cracking application. It works well with web applications and can use different protocols, as well as not just cracking passwords but also usernames. You can supply two different wordlist files, usernames and passwords as inputs and it can crack both simultaneously.

In this task, you will use Medusa to crack the username and password on the bWAPP web application. To do this, perform the following steps:

Step 1

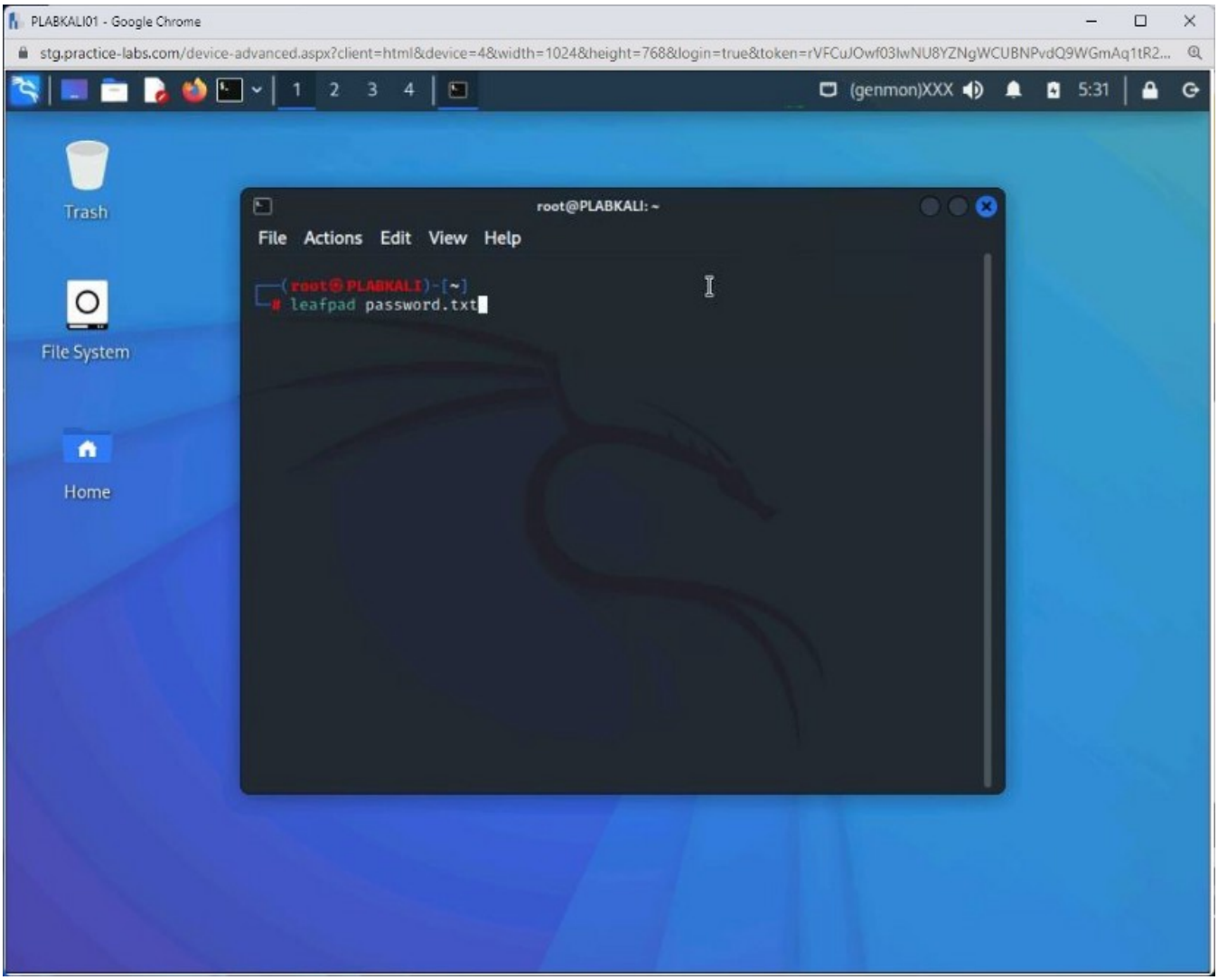
Connect to **PLABKALI01**. Open a new terminal window.

You have an option to use a pre-defined wordlist. There are several wordlists available that have grown into Gigabytes of size. You can also download the wordlists from the Internet. An alternative is to create a small wordlist manually, which you will do now.

This file will contain the keywords that will be used for guessing the password. In the command prompt window, type the following command:

```
leafpad password.txt
```

Press **Enter**.



Step 2

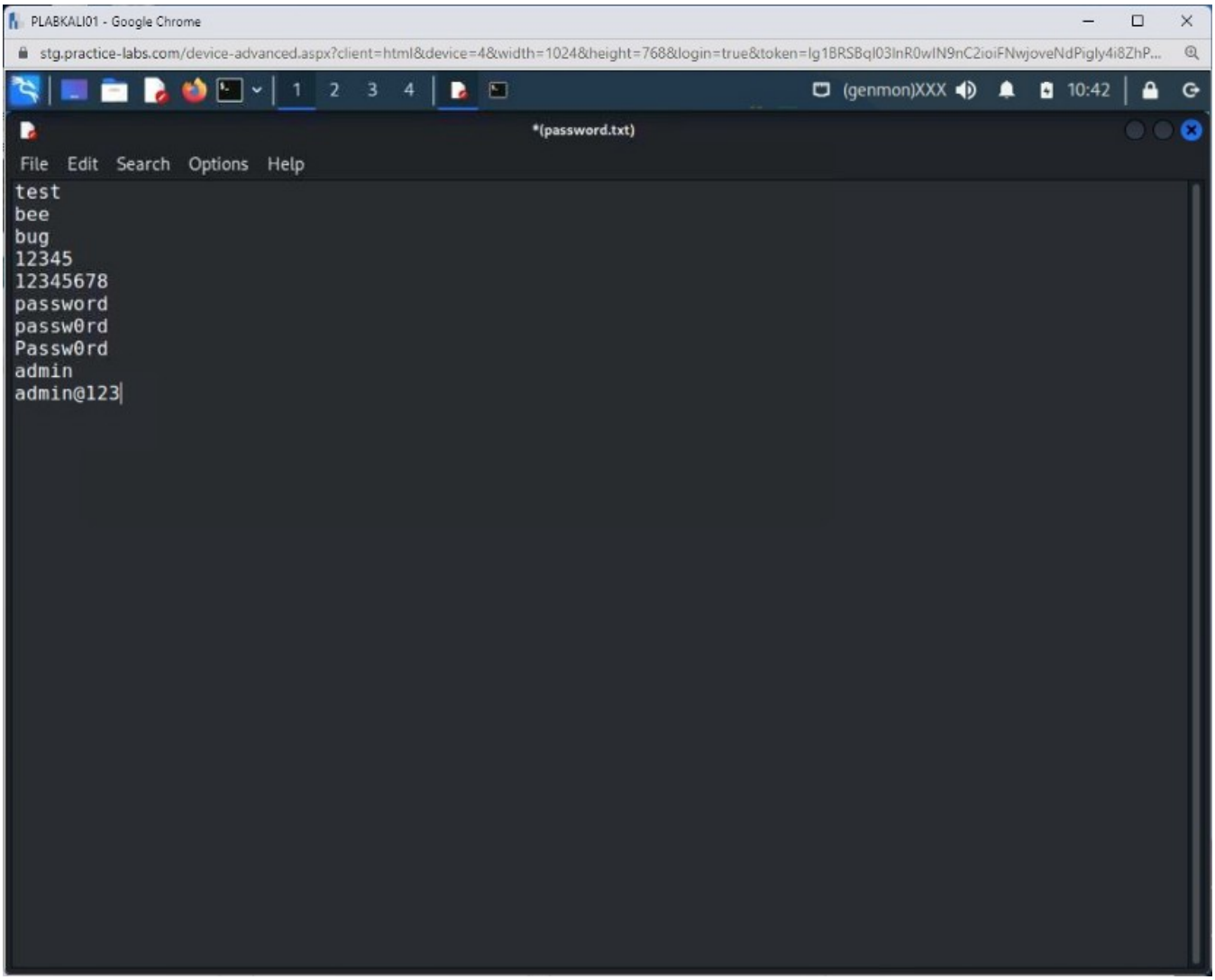
Leafpad opens with a file named **(password.txt**).

Type the following words:

```
test
bee
bug
12345
```

```
12345678
password
passw0rd
Passw0rd
admin
admin@123
```

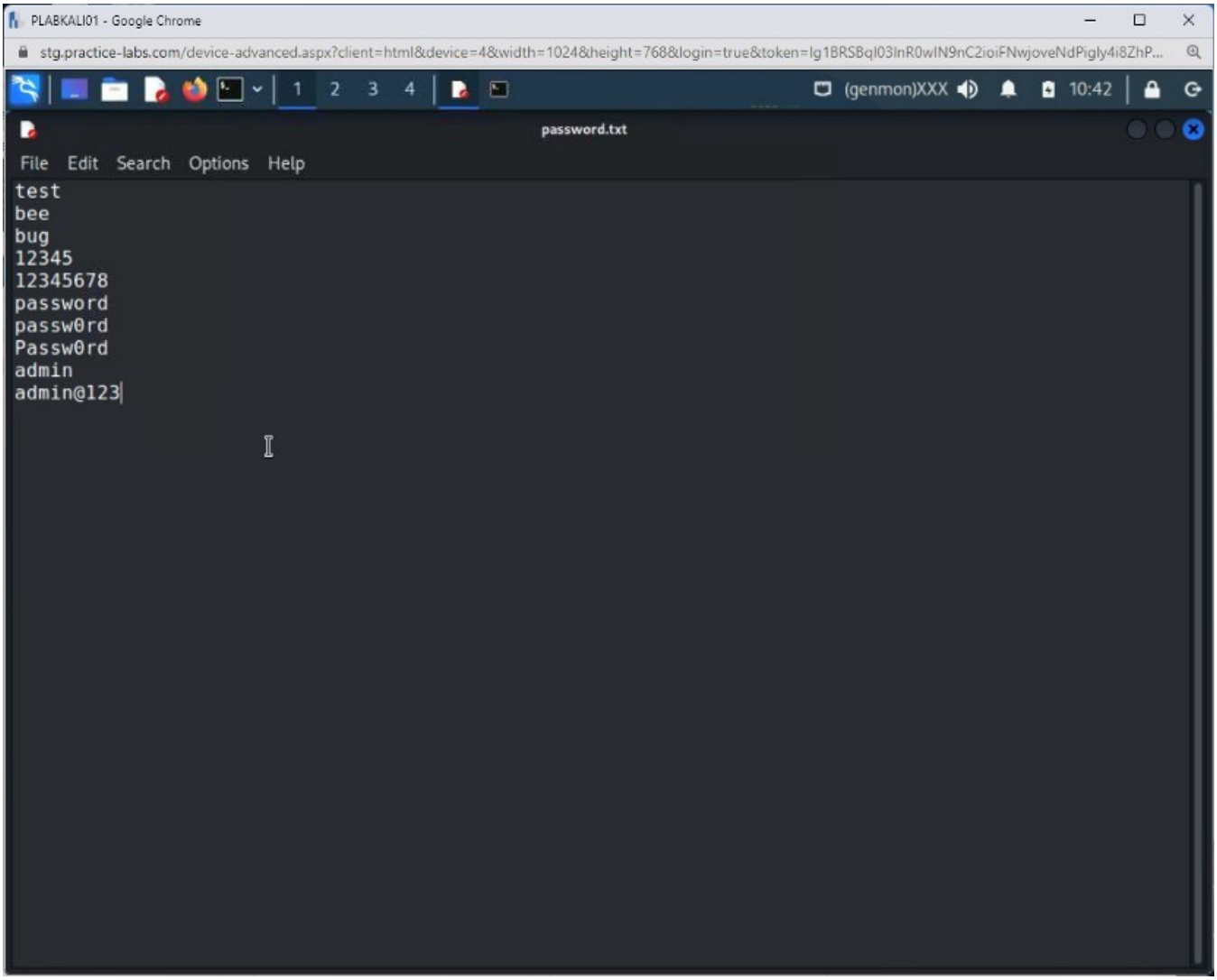
Press **Enter** after each word except the last one.



Step 3

Press **Ctrl + S** to save the file.

Close the **password.txt** file.

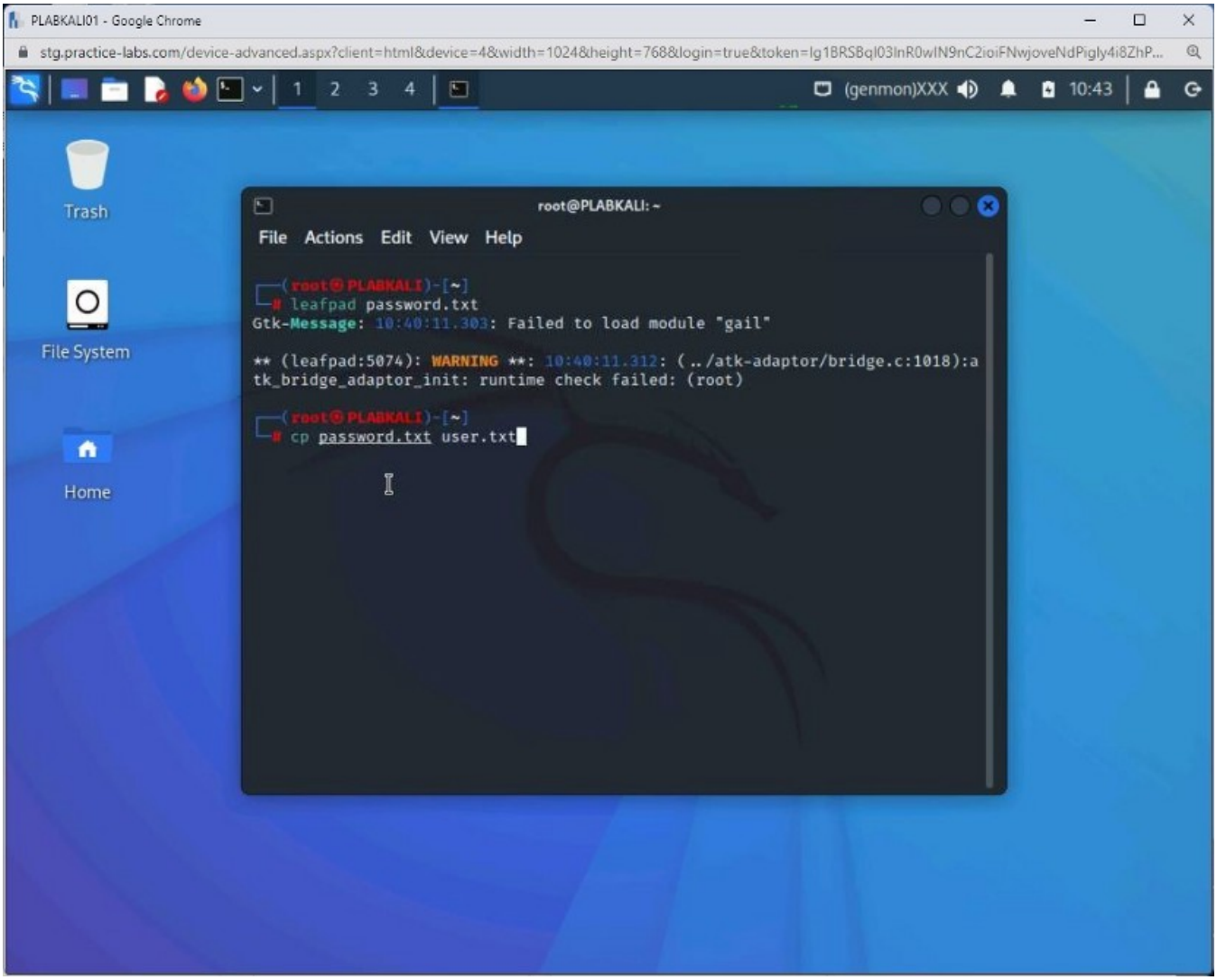


Step 4

Let's make a copy of this file as the usernames file. Type the following command:

```
cp password.txt user.txt
```

Press **Enter**.

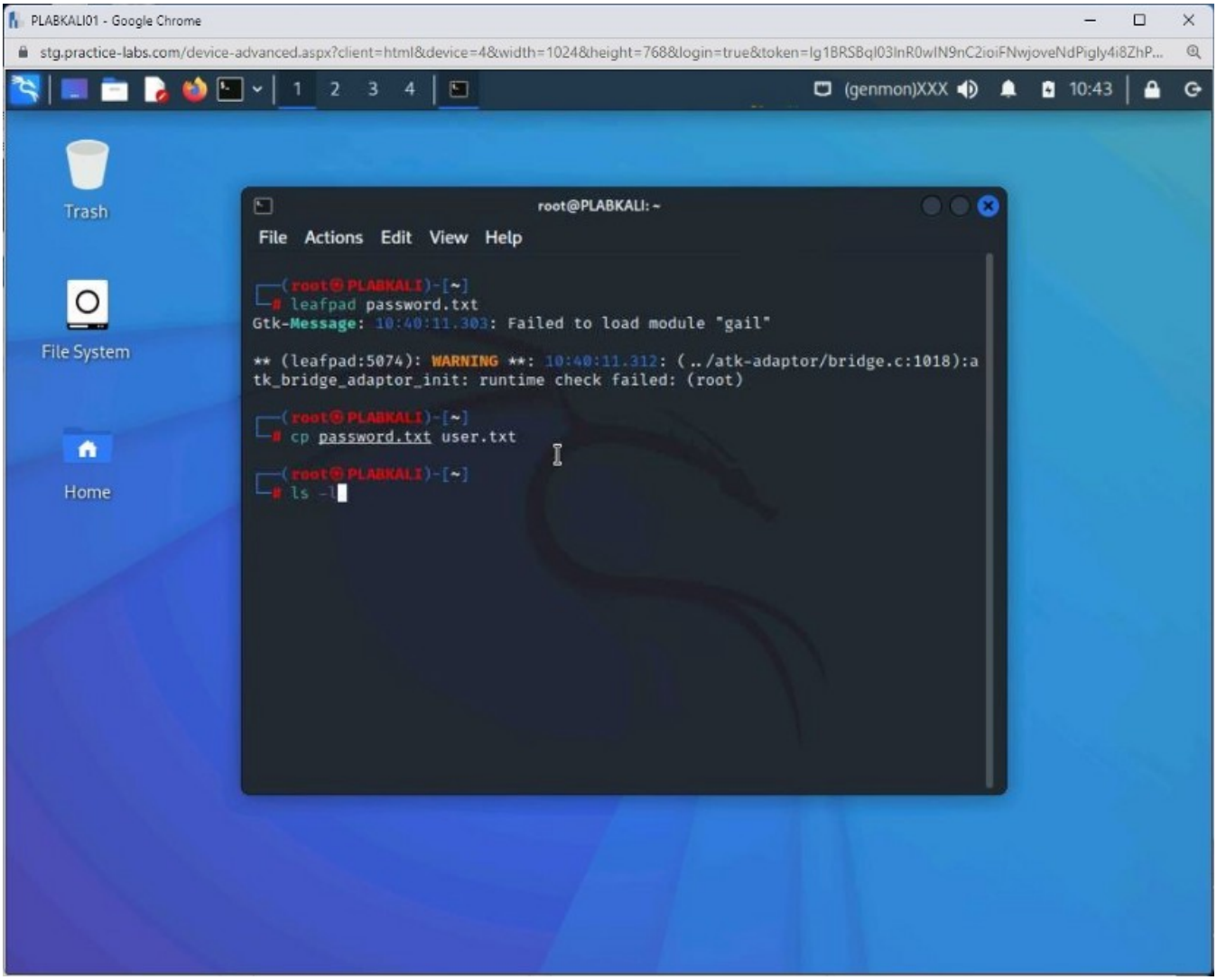


Step 5

The **cp** command does not return any output. To verify you have two files, **plab.txt** and user.txt, type the following command:

```
ls -l
```

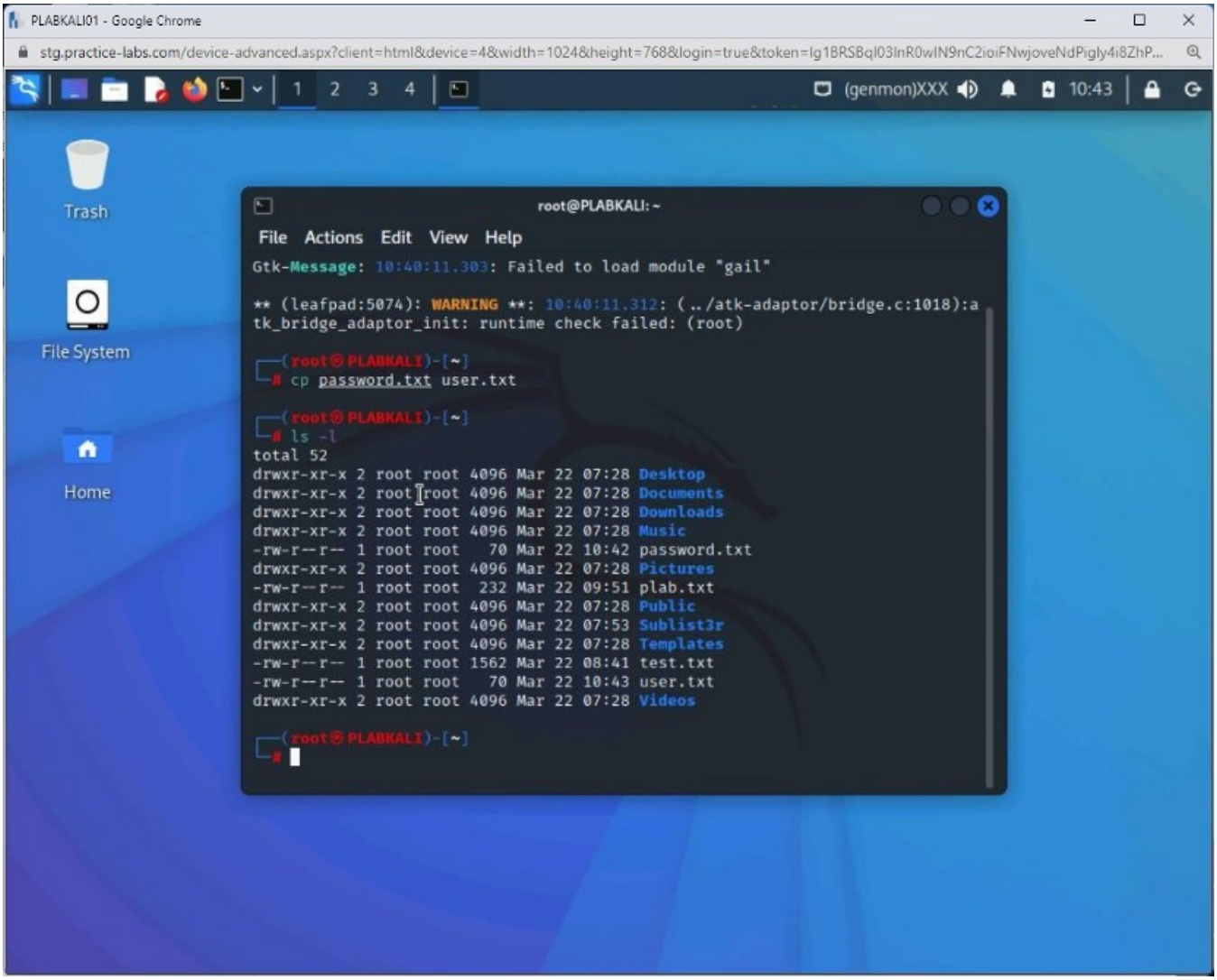
Press **Enter**.



Step 6

The file listing contains both files.

***Note**: Depending on which modules have been completed prior to this one, the plab.txt may or may not already exist.*



Step 7

Clear the screen by entering the following command:

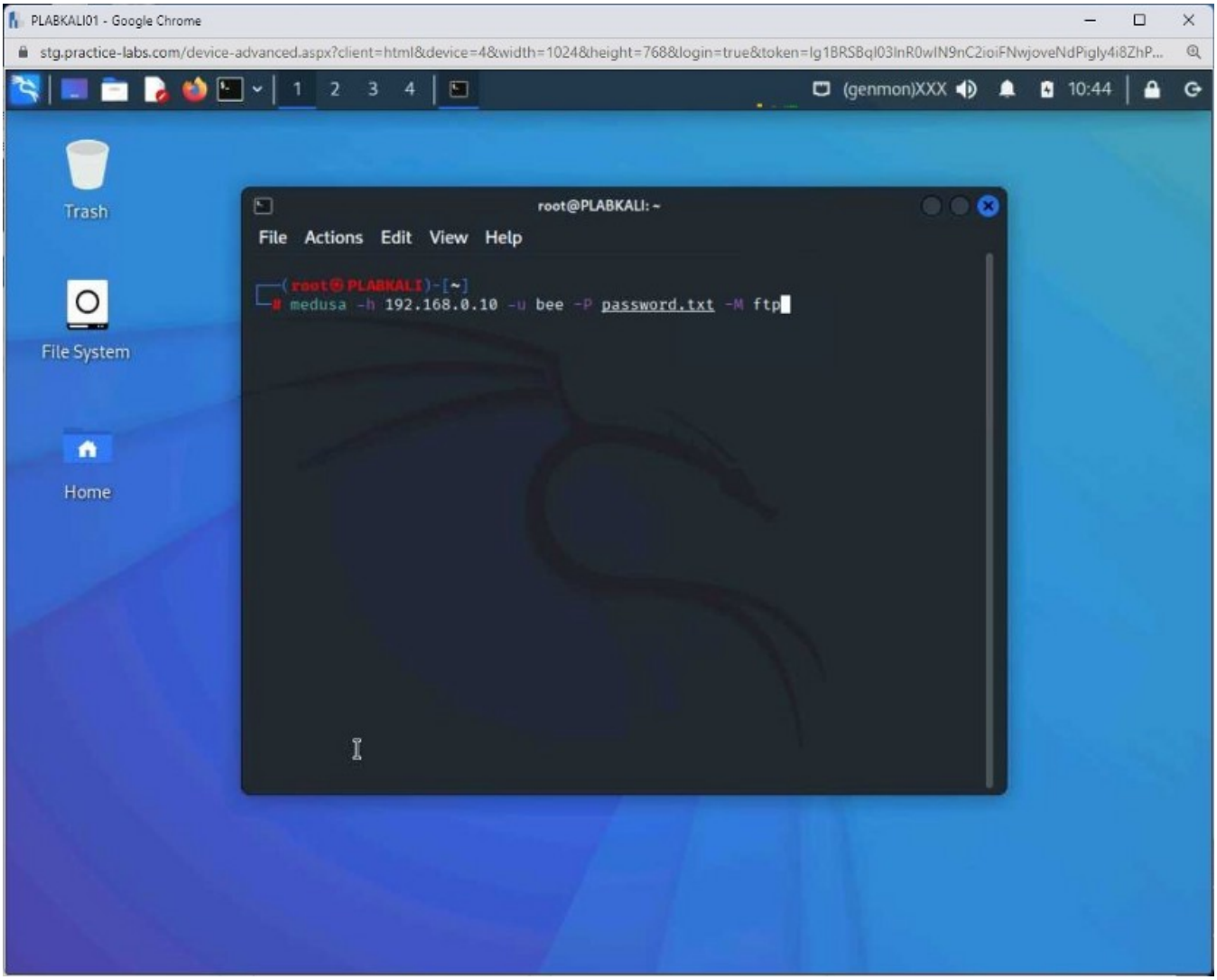
```
clear
```

You will now use the **plab.txt** as the password wordlist and attempt to guess the password for the username **bee**. You will attempt to break the password of the FTP account running on the webserver, **192.168.0.10**. Type the following command:

Note: *The command below uses the following parameters:*
*-**h**: IP address of the target system*
*-**u**: Username. If -U is used, you need to specify the username wordlist.*
*-**P**: Password wordlist. If -p is used, then you can specify a single password.*
*-**M**: Module used for cracking the password*

```
medusa -h 192.168.0.10 -u bee -P password.txt -M ftp
```

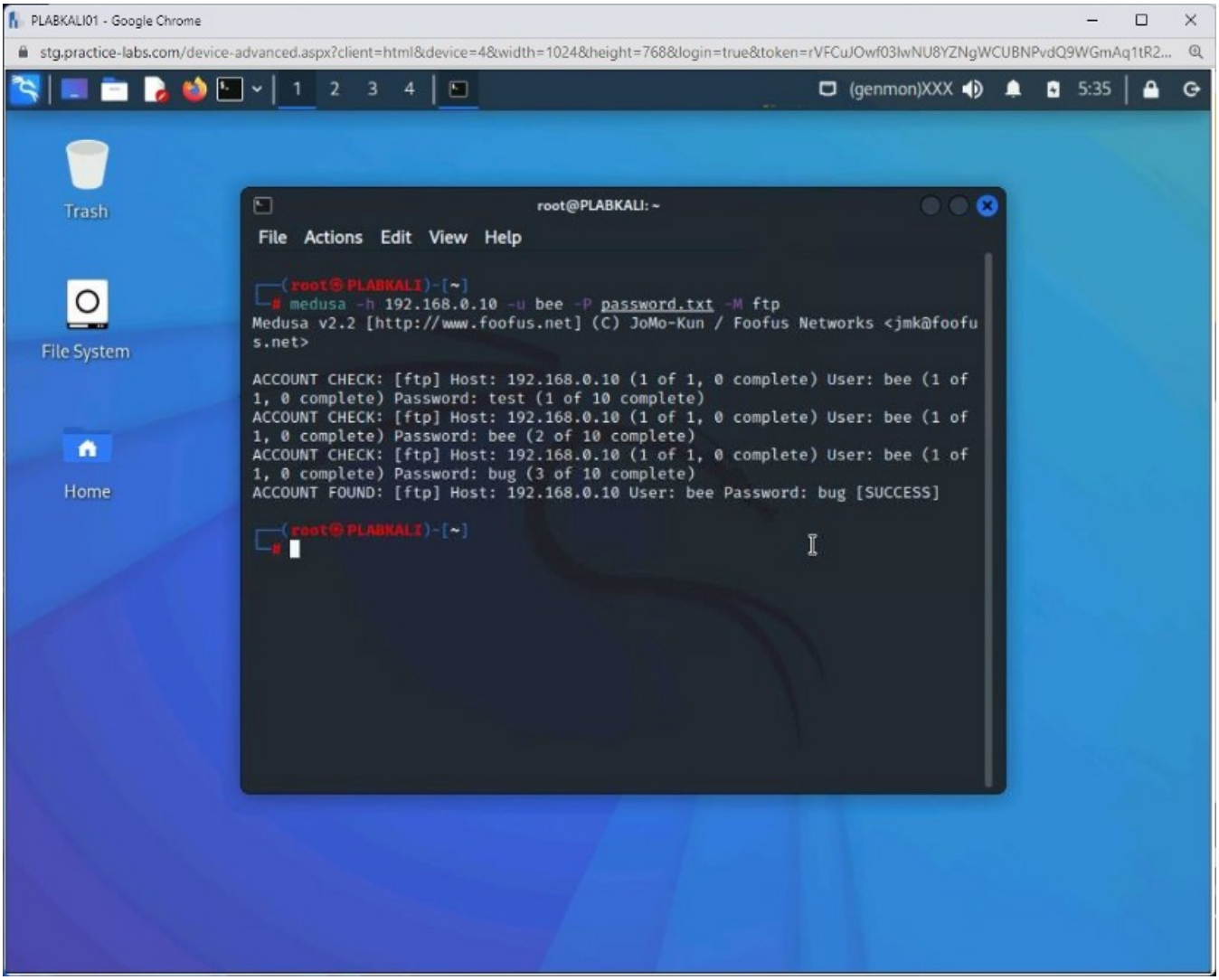
Press **Enter**.



Step 8

The password cracking process starts. It uses each word from the **password.txt** against the username **bee**.

Finally, the correct password is found. The last statement lists the password as **bug**. Notice that after the correct password is found, Medusa stops the process.

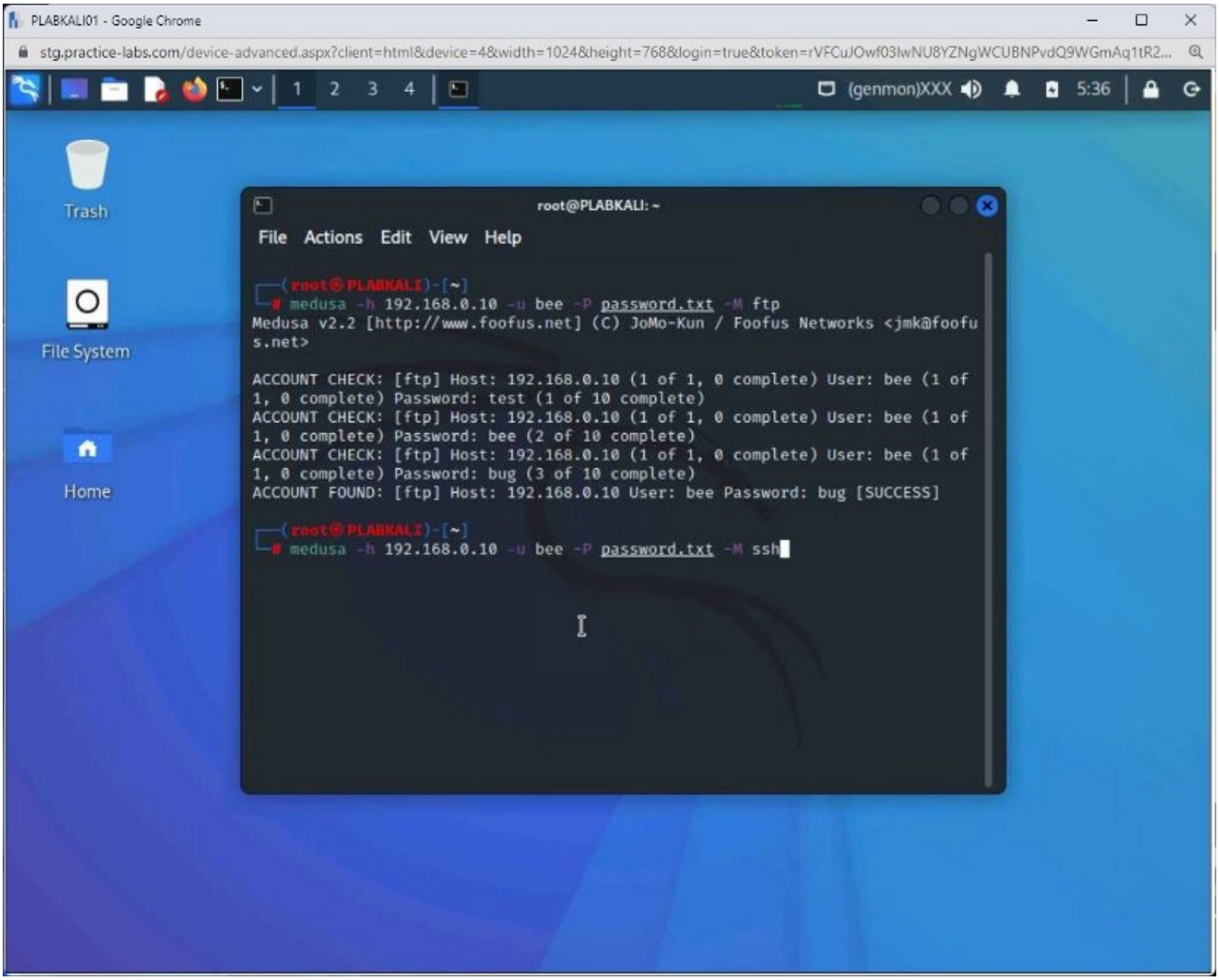


Step 9

Let's attempt to crack the password for the **SSH** module. Type the following command:

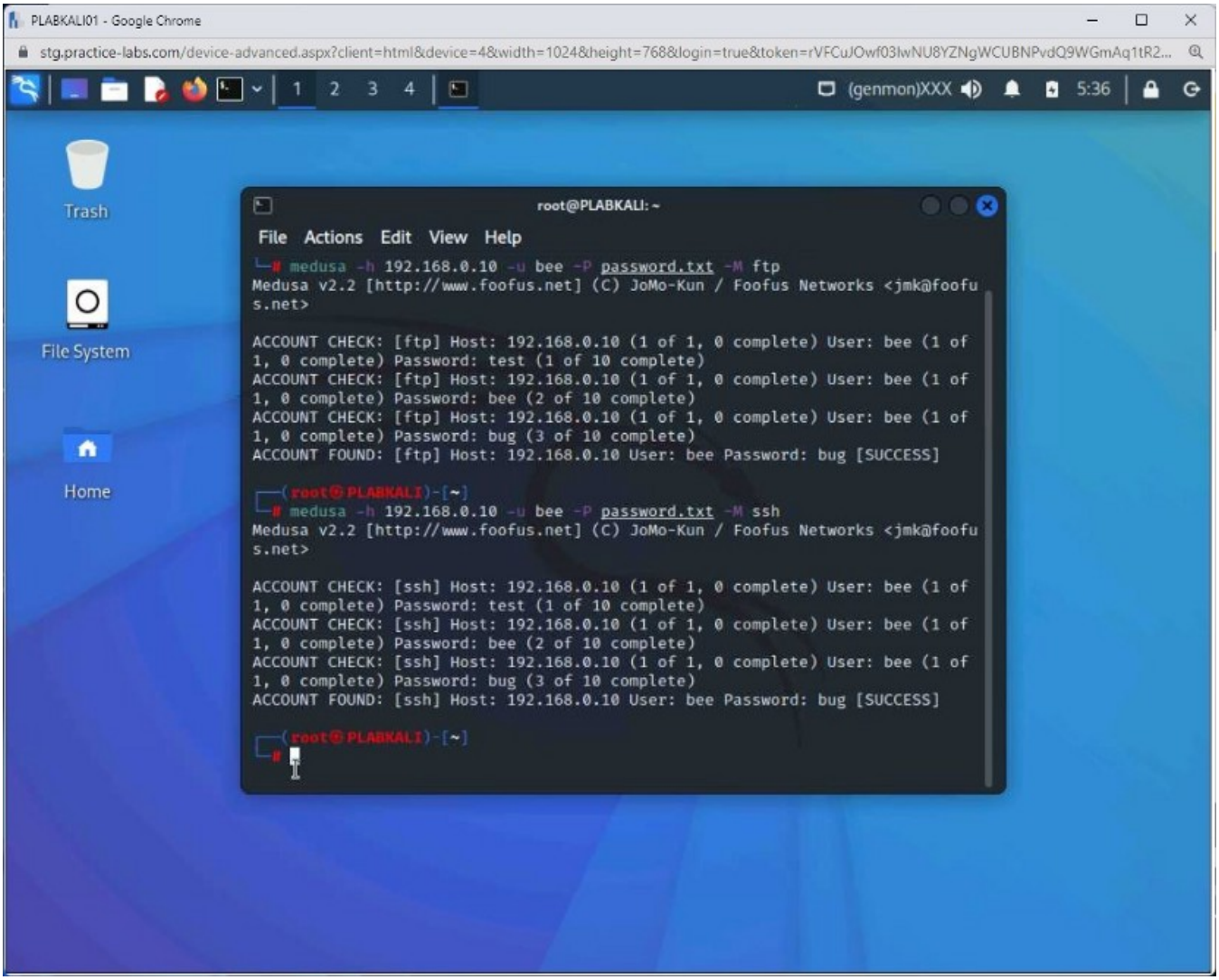
```
medusa -h 192.168.0.10 -u bee -P password.txt -M ssh
```

Press **Enter**.



Step 10

Like the FTP module, the password for SSH is also cracked for the username **bee**.



Step 11

Clear the screen by entering the following command:

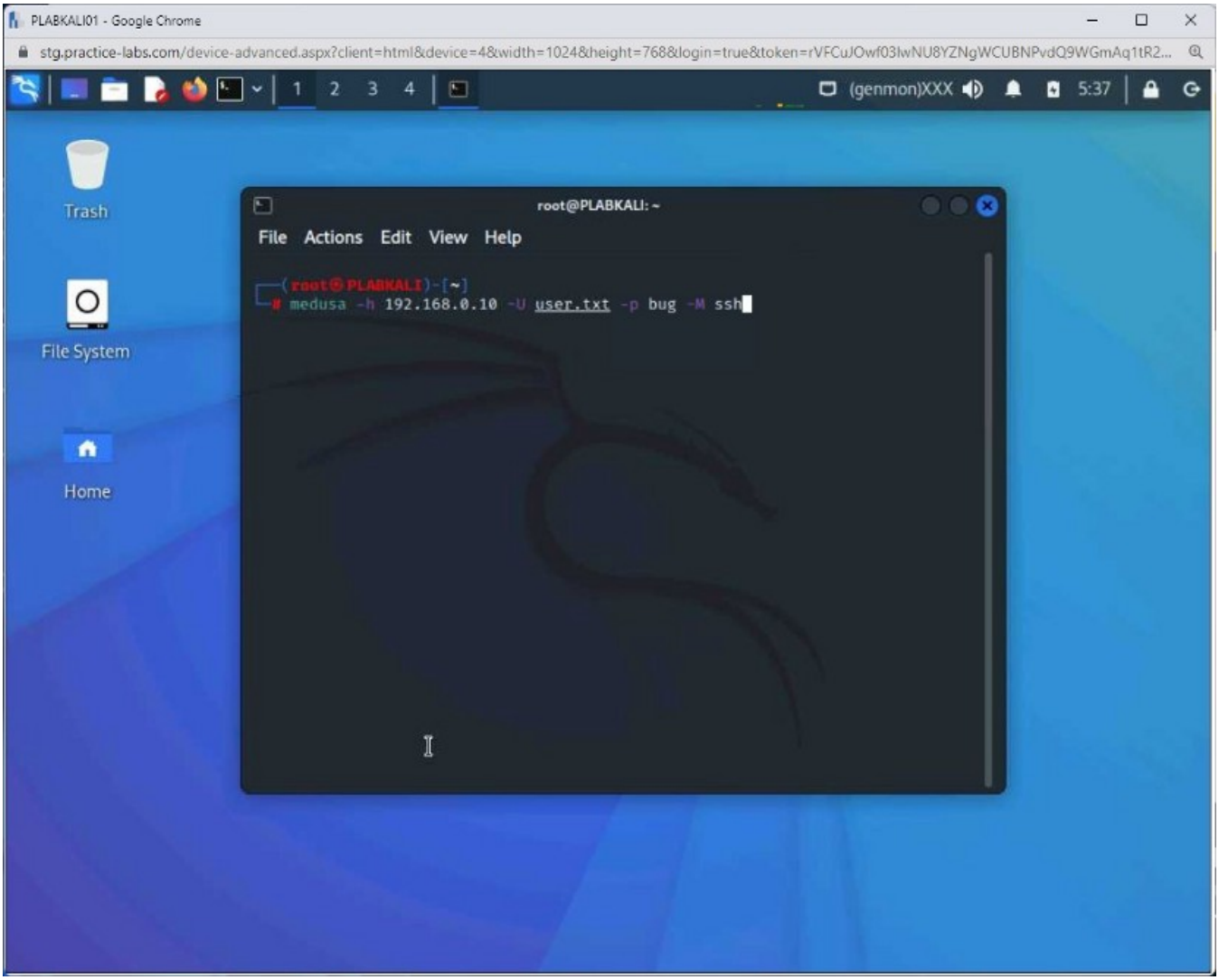
```
clear
```

Let’s find the username by providing the password as an input.

Type the following command:

```
medusa -h 192.168.0.10 -U user.txt -p bug -M ssh
```

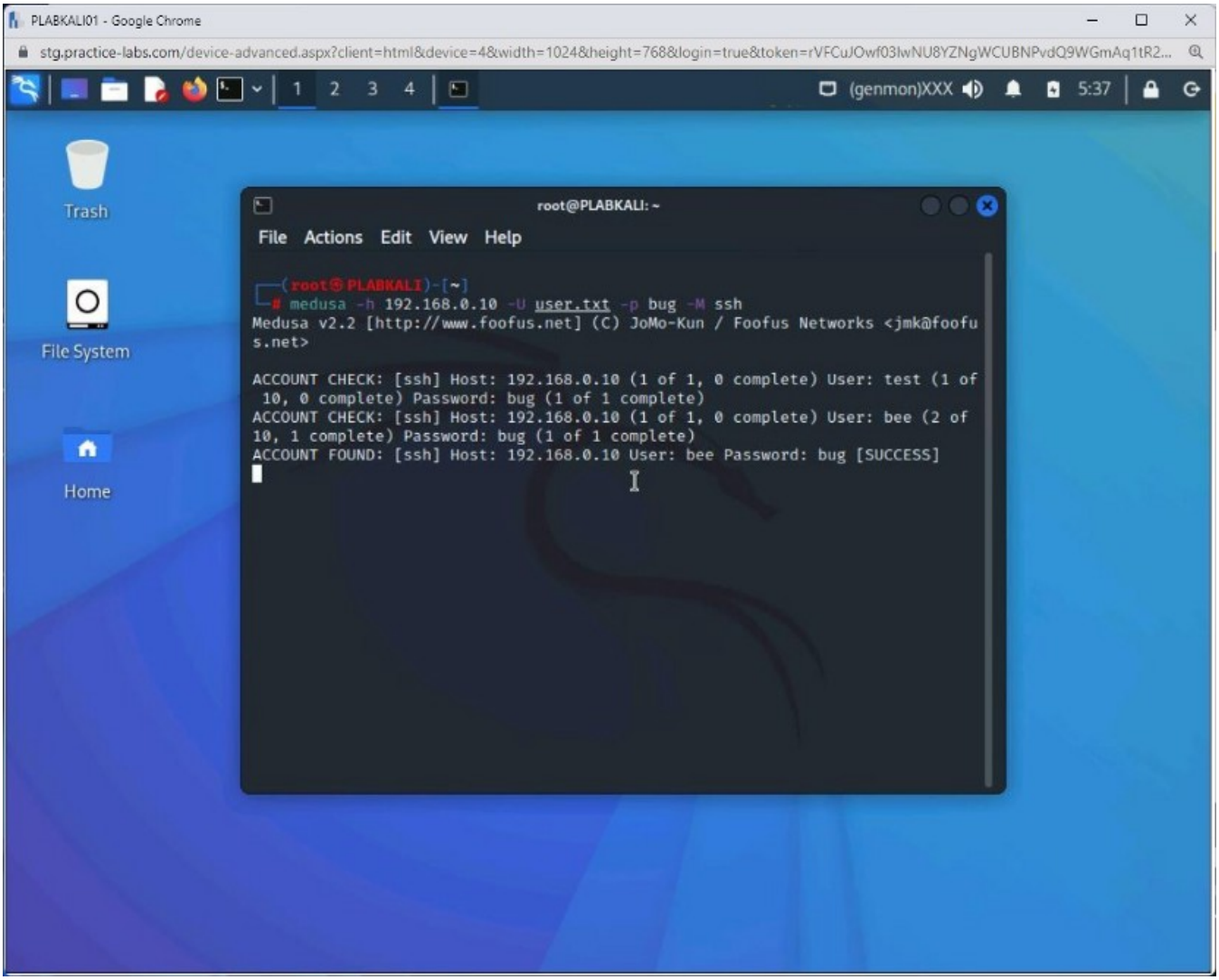
Press **Enter**.



Step 12

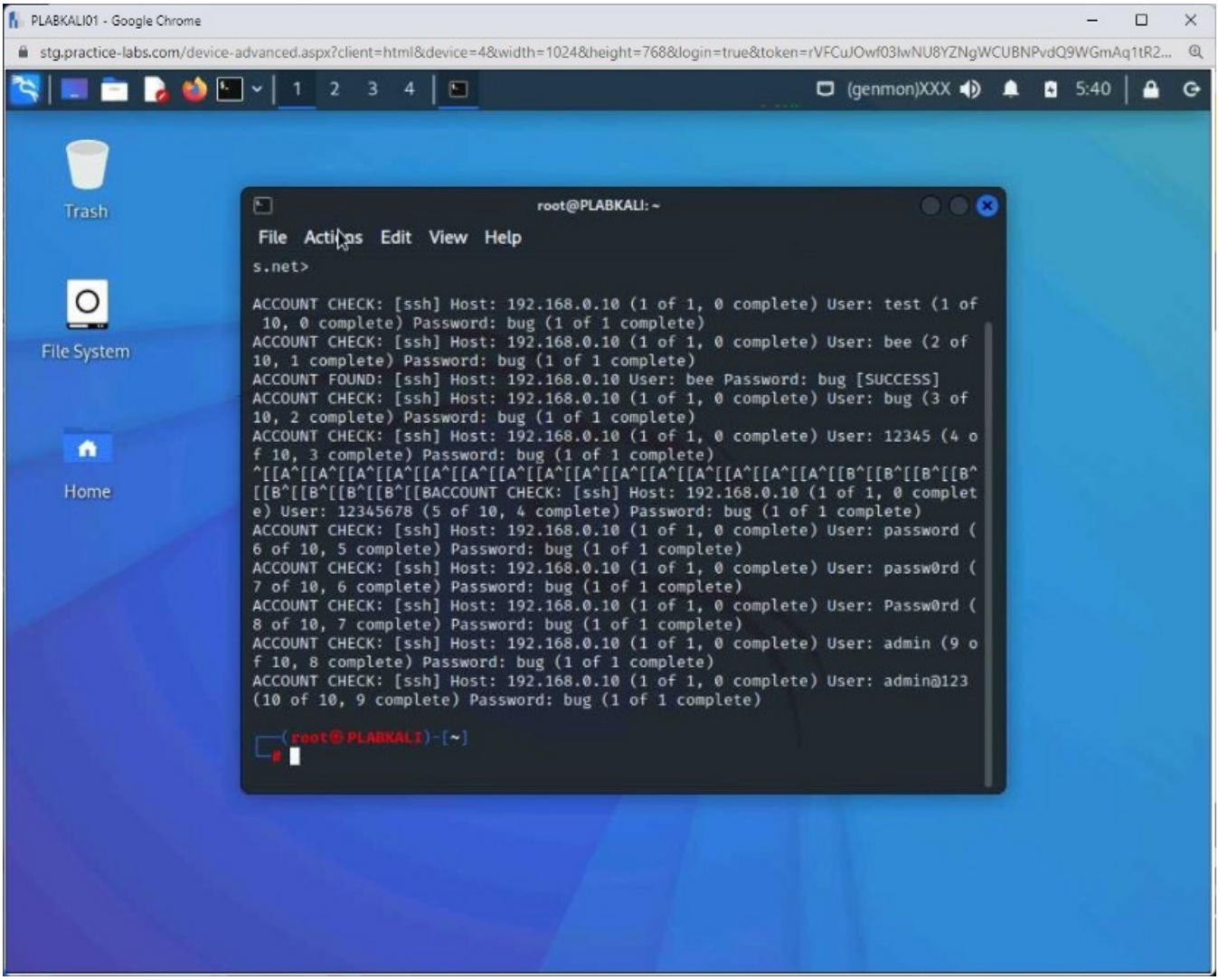
The username cracking process starts. Notice the statement with **[SUCCESS]**, meaning it has found one username.

However, unlike password cracking, the username cracking process does not stop and attempts to find more usernames from the given wordlist. It will run through all the usernames given in the wordlist.



Step 13

The username cracking process stops after running through the usernames in the wordlist.



Step 14

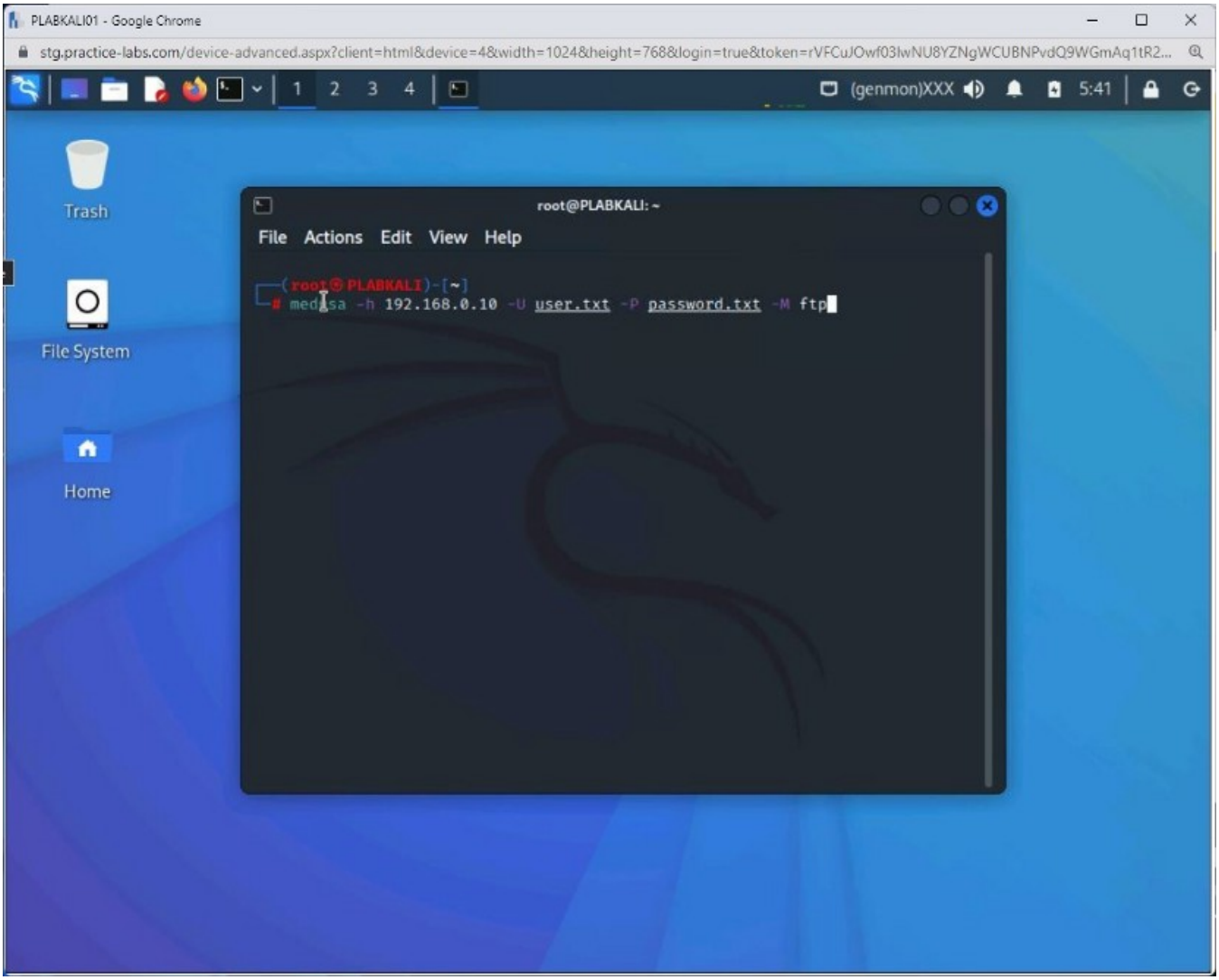
Clear the screen by entering the following command:

```
clear
```

Let's now attempt to use the username and password wordlists to find the username and the password. To do this, type the following command:

```
medusa -h 192.168.0.10 -U user.txt -P password.txt -M ftp
```

Press **Enter**.

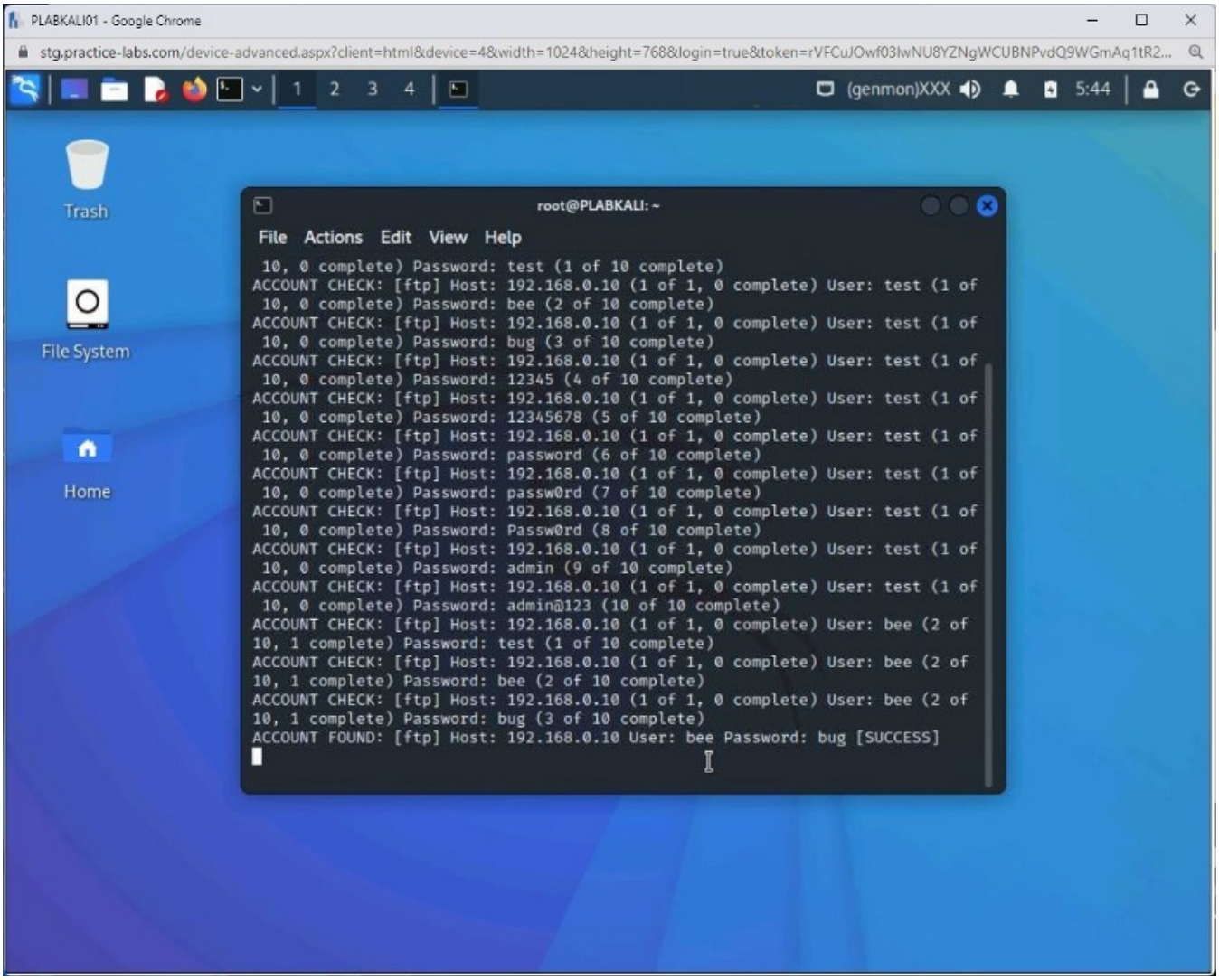


Step 15

The username and password cracking process start. Each username will be run against each password in the **password.txt**.

Note: *The username and password cracking process will take a while to complete.*

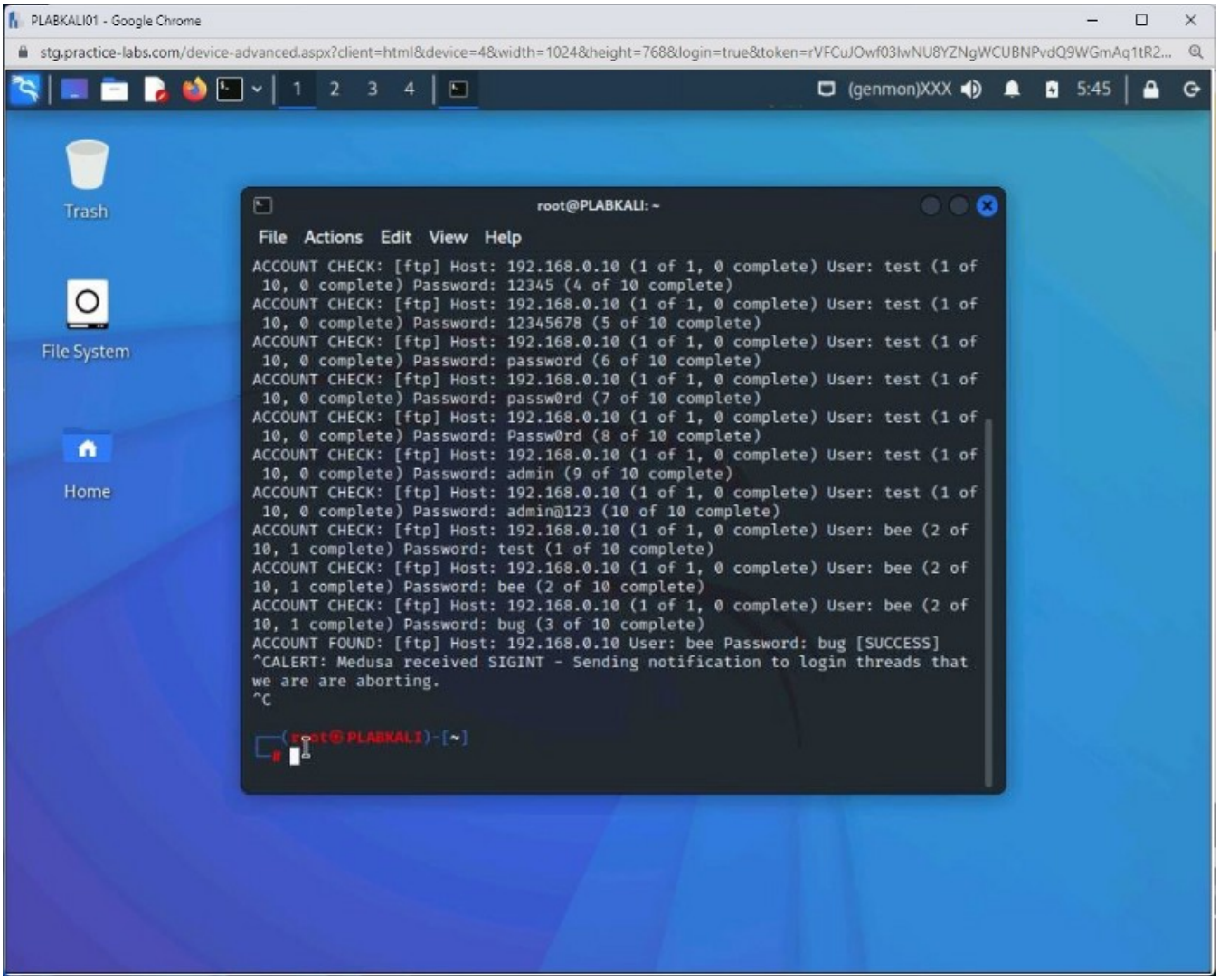
Finally, it can match the username with the correct password.



Step 16

You have found a correct combination of username and password (**bee** and **bug**), and therefore, it is safe to abort the process.

Press **Ctrl + C** to break the process.



Keep the terminal window open.

Task 3 — Perform Broken Authentication Attacks

Authentication is a process that confirms a user's identity using a username and password. The server or the web application validates the user's identity. In a web application scenario, the authentication process is as follows:

1. On a login form, a user enters their login credentials, typically via username and password.
2. After a user submits the credentials, they are then sent to the application where they are verified, and then a session is created and stored in a database.
3. A cookie is sent to the user's system.
4. When a user makes a subsequent request, the session ID is verified with the one stored in the database. The browser at the client end stores and sends the token as a

cookie to the server. If both the session ID values match, the user's request is processed. However, if the values do not match, the request is not processed further.

5. After performing certain tasks, the session is destroyed when a user logs out from the application, the client, and the server.

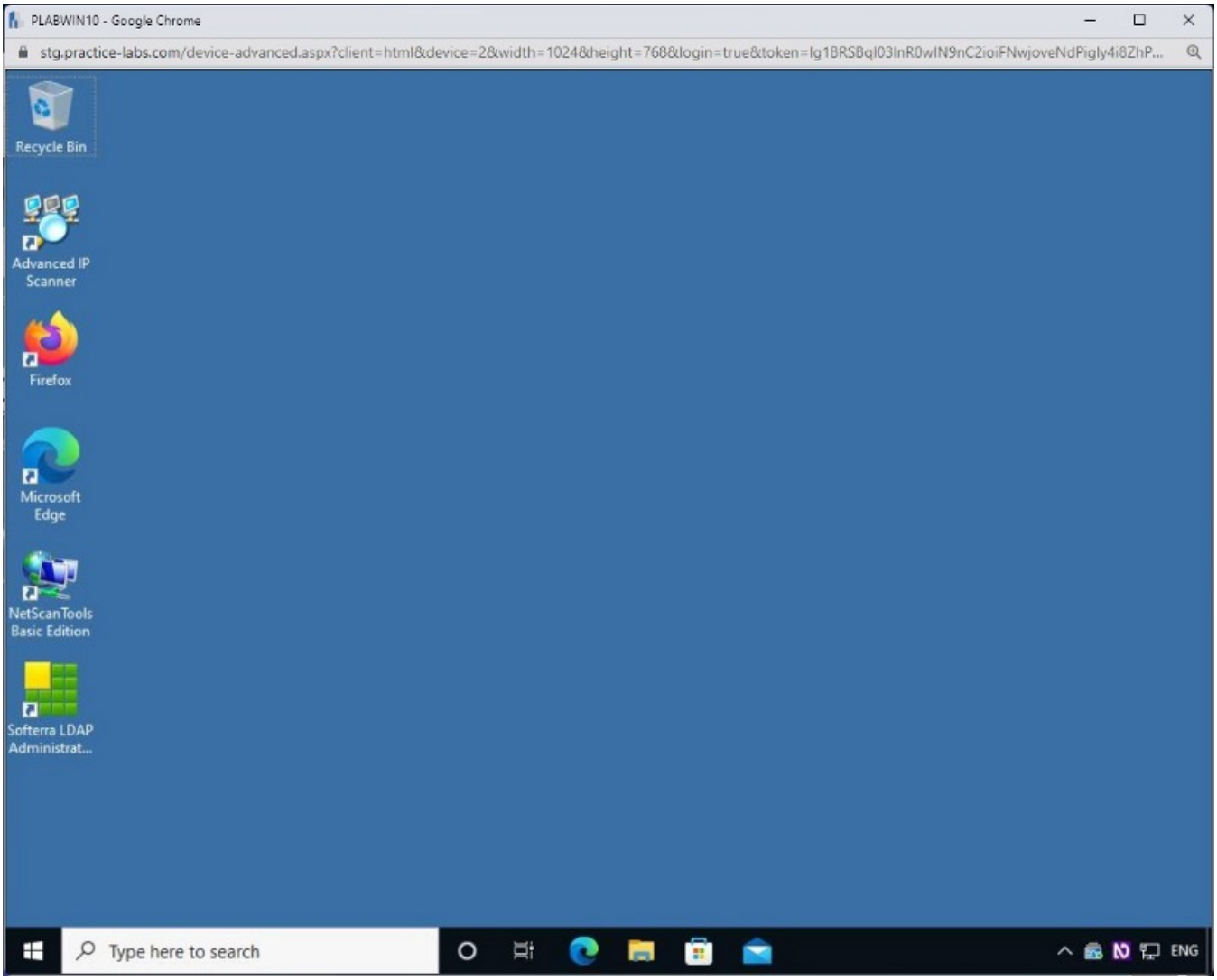
These methods are also known as Access Control attacks, in that they allow privilege escalation. Using the same method, an attacker can perform the following attacks:

- Parameter-based Access Control
- Referer-based Access Control
- Location-based Access Control

In this task, you will learn to perform broken authentication attacks. To do this, perform the following steps:

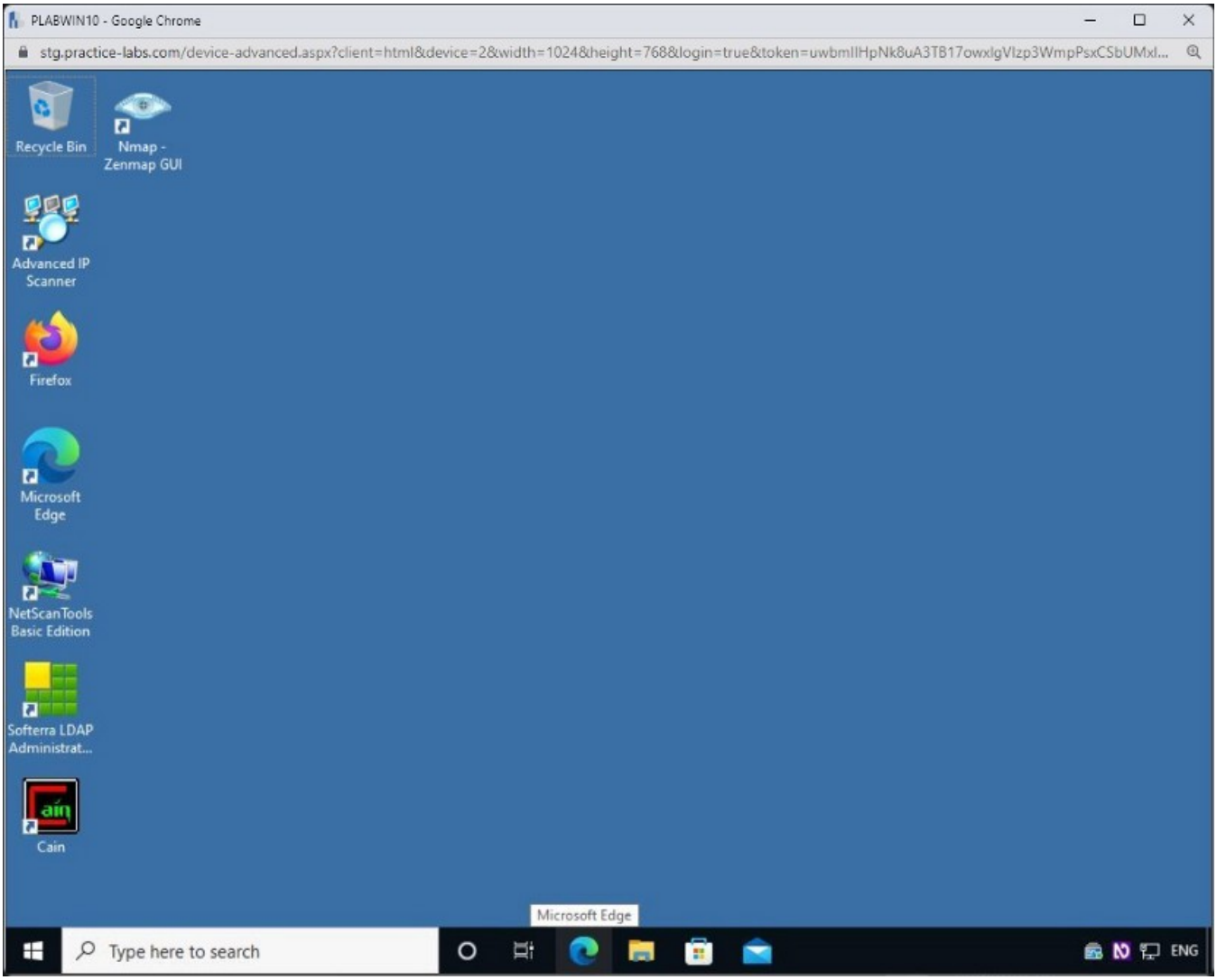
Step 1

Connect to **PLABWIN10**. The desktop is displayed.



Step 2

Open **Microsoft Edge** by clicking on the icon on the taskbar.

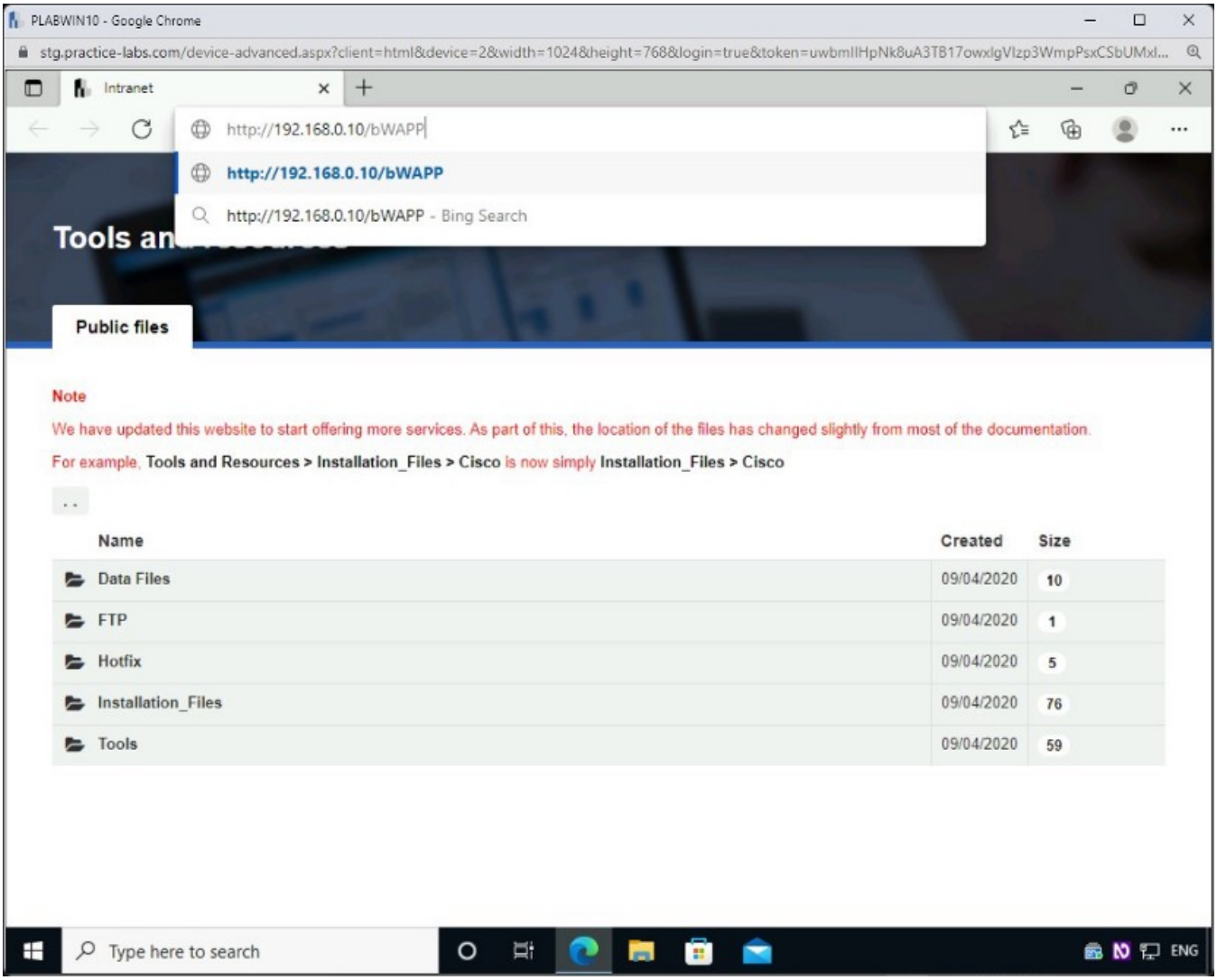


Step 3

The **Microsoft Edge** window is displayed. In the address bar, type the following URL:

```
http://192.168.0.10/bWAPP
```

Press **Enter**.



Step 4

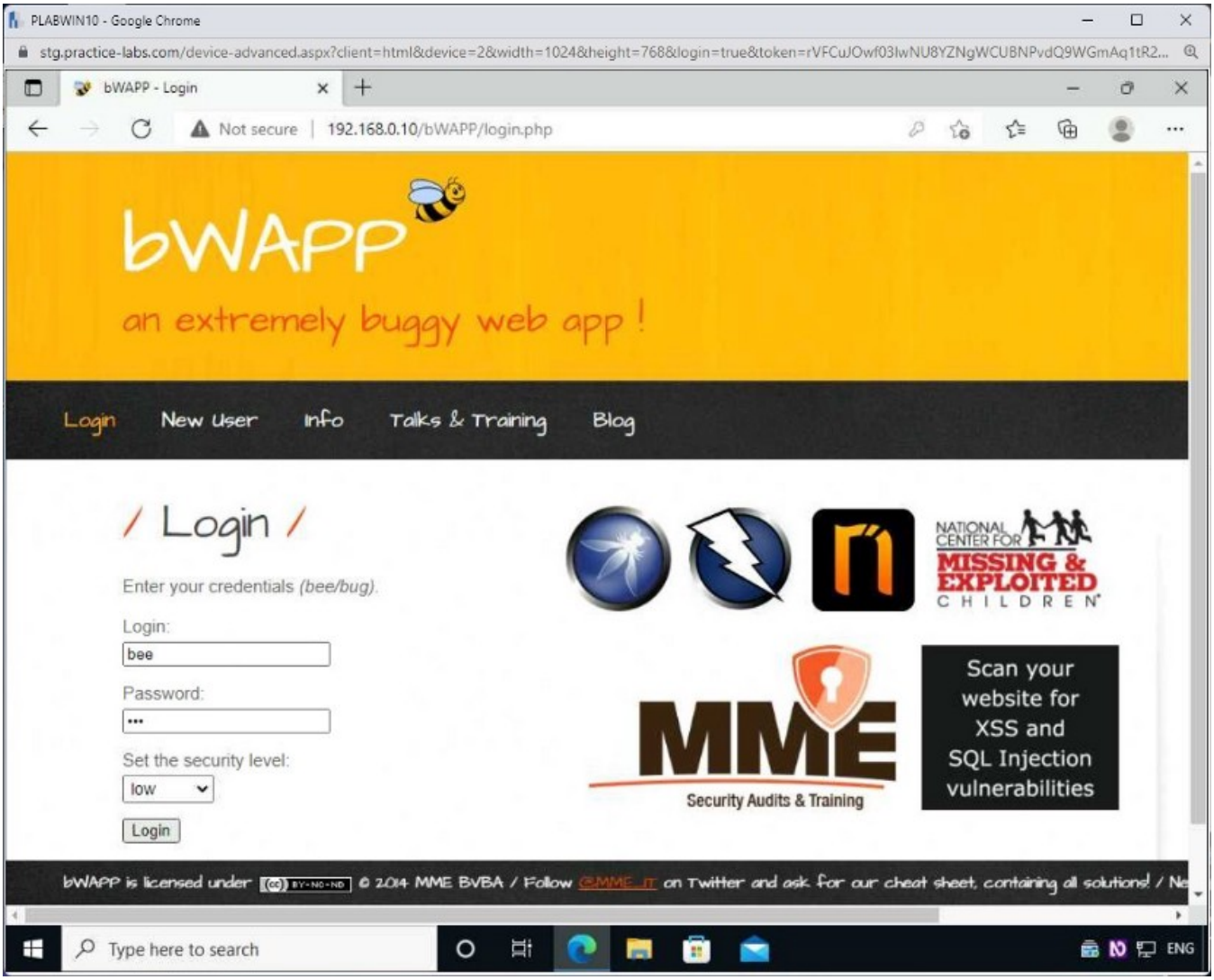
The login page of the **bWAPP** web application is displayed. In the **Login** text box, type the following username:

```
bee
```

In the **Password** text box, type the following password:

bug

Click **Login**.



Step 5

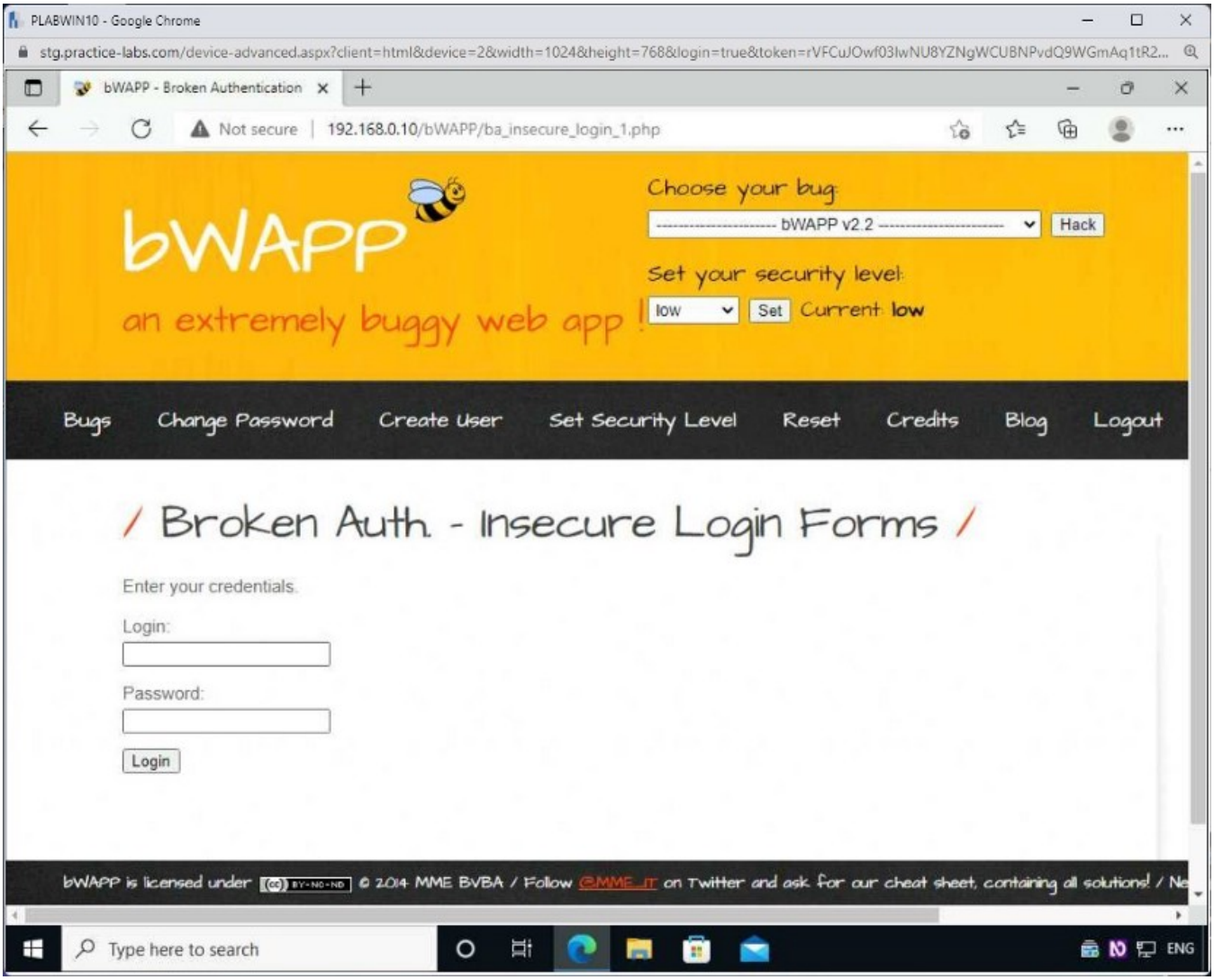
A notification bar is displayed about remembering the password. Click **Not for this site**.

Step 6

From the **Choose your bug:** drop-down, select **Broken Authentication — Insecure Login Forms,** and click **Hack**.

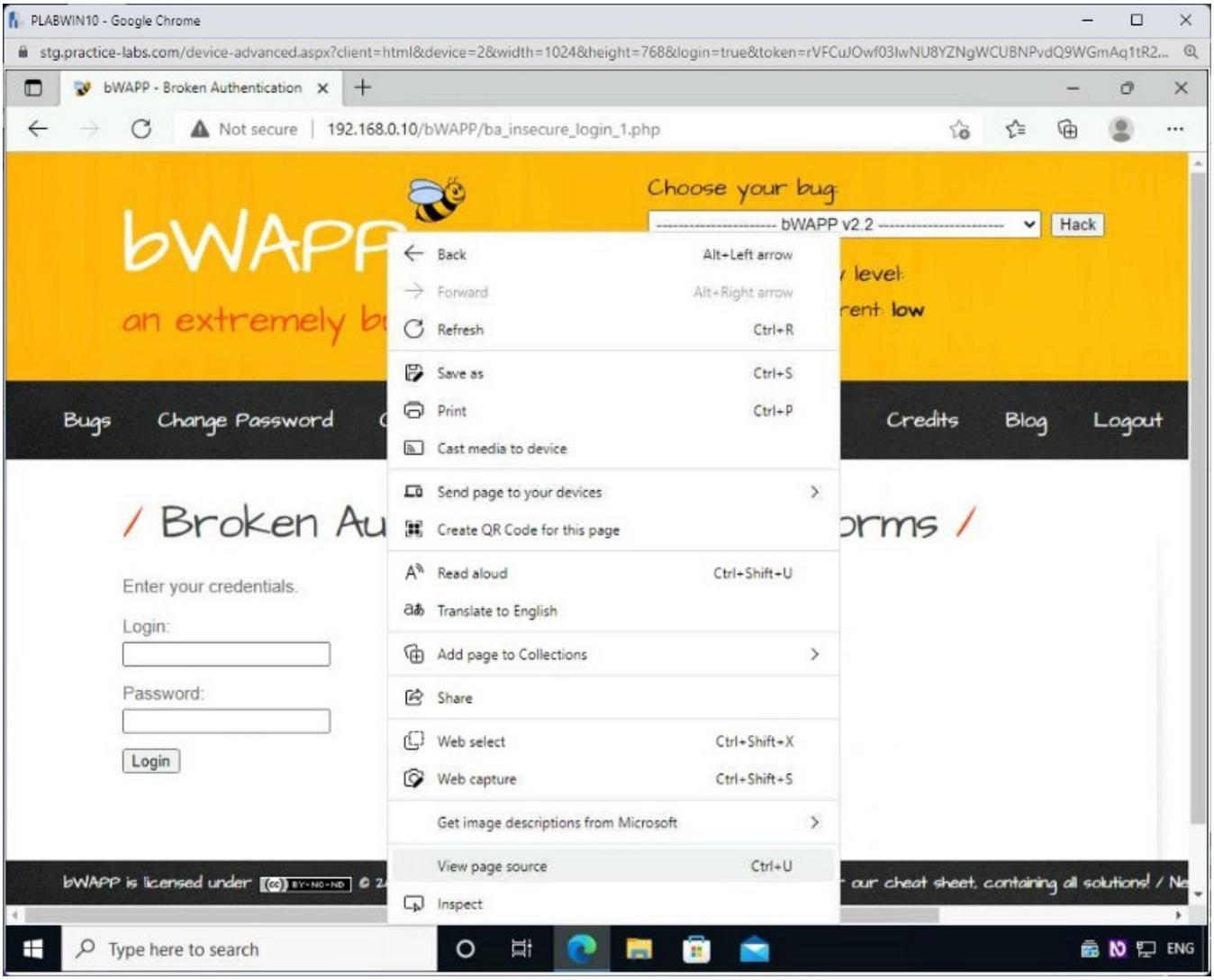
Step 7

The **login** page is displayed.



Step 8

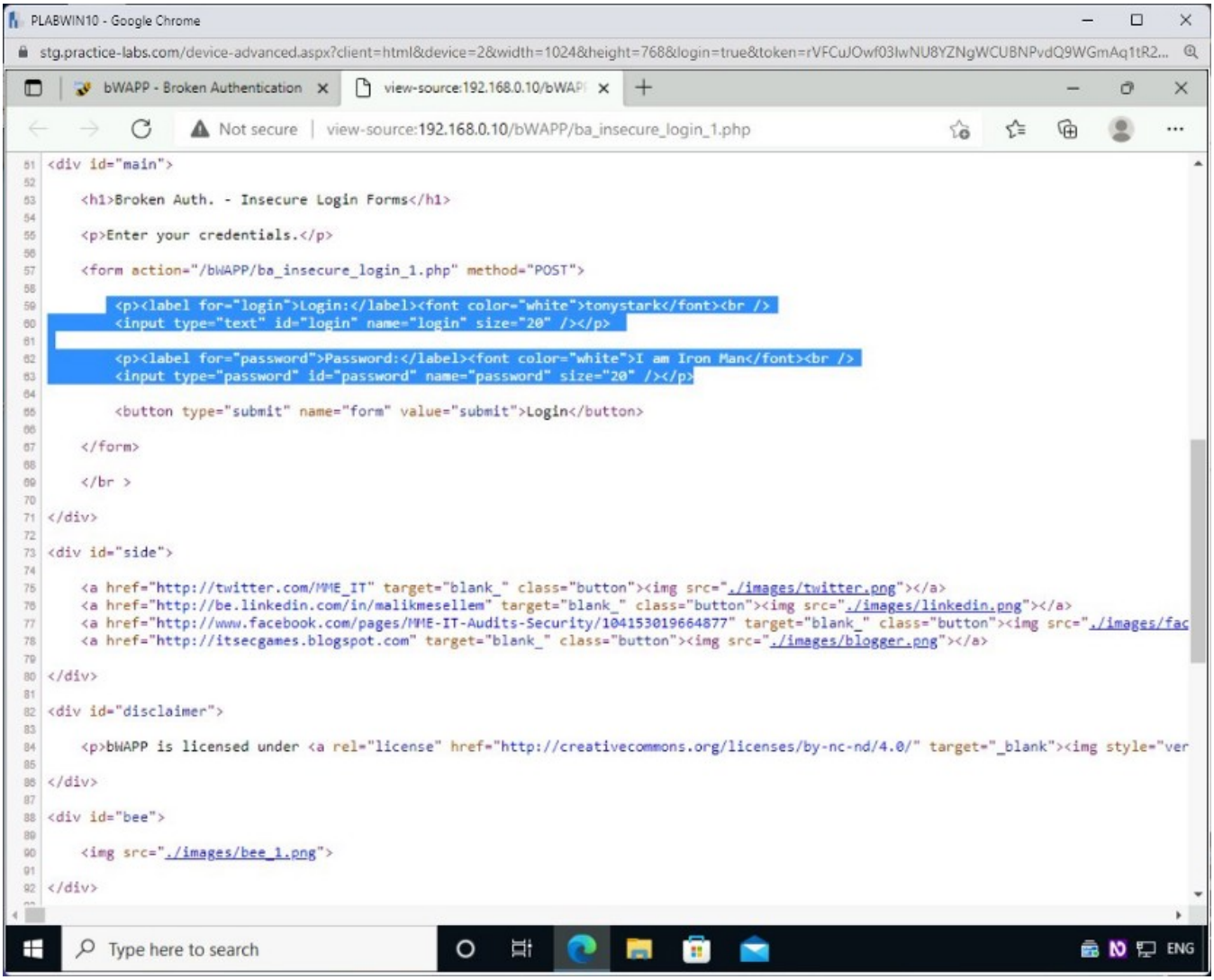
Several times, the developers hard-code the login information in the login form. It is quite easy to extract login information. To do this, right-click anywhere on the login form and select **View source**.



Step 9

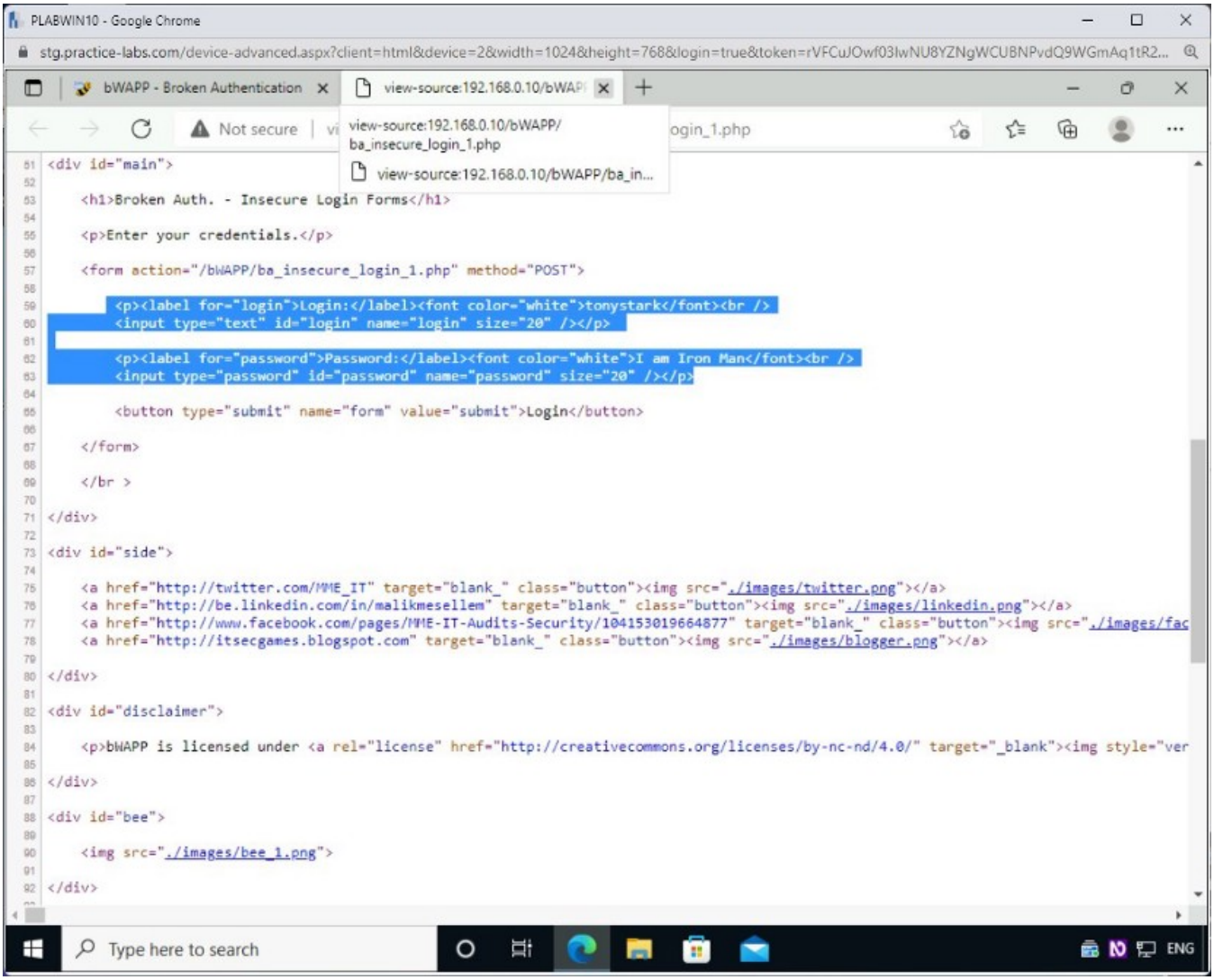
A **Debugger** window opens in the bottom section of the webpage and displays the source code for the login page. Review the code from lines **59** to **63**.

Notice the username and password. It displays the login name as **tonystark** and password as **I am Iron Man**.



Step 10

Close the **view-source** tab.

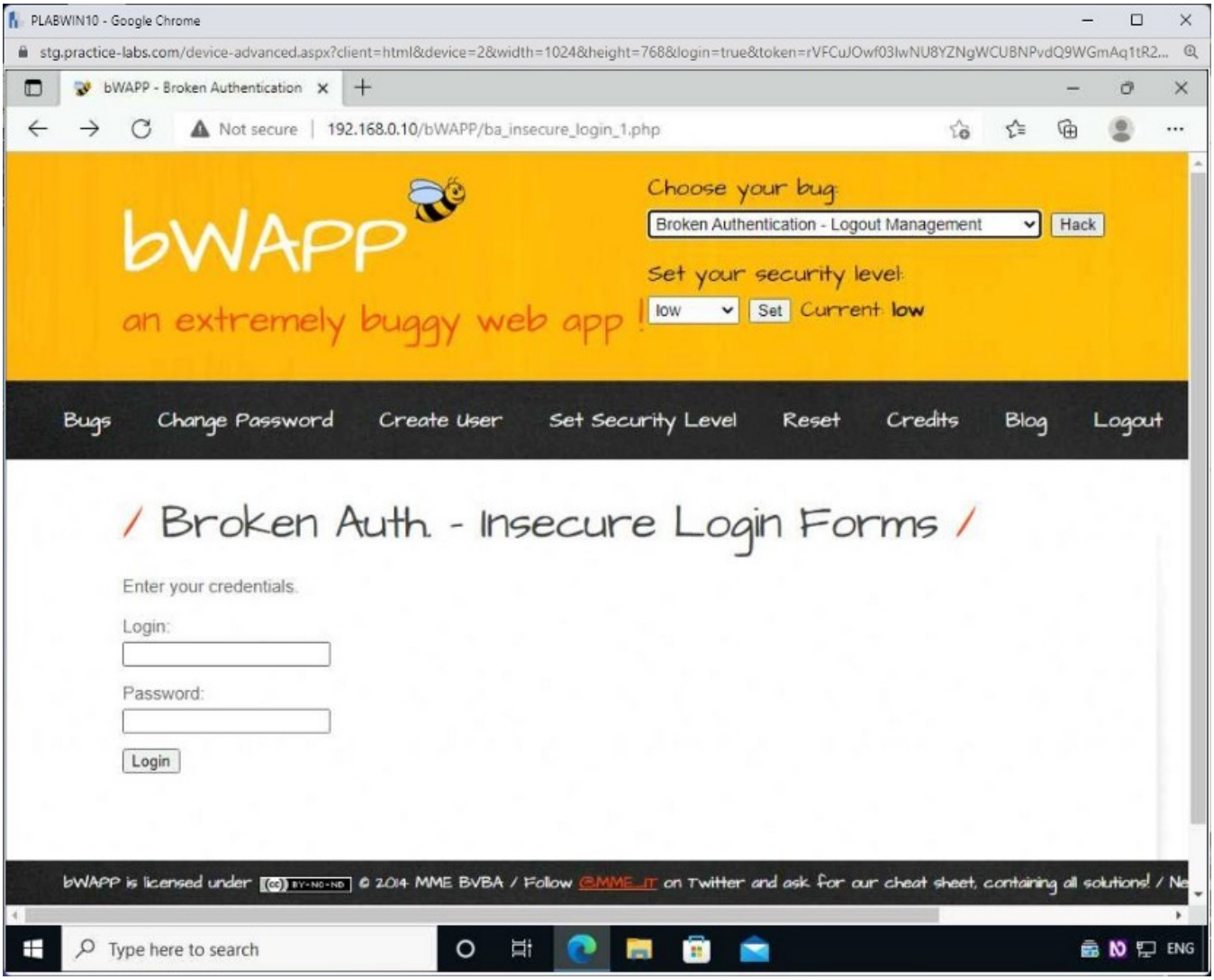


Step 11

Logout management is one of the common vulnerabilities in applications. Even if the user logs out, the user session is still live. One of the common methods is to click the Back button on the browser to get back to the same session after logging out.

Let's perform a logout management attack on this application.

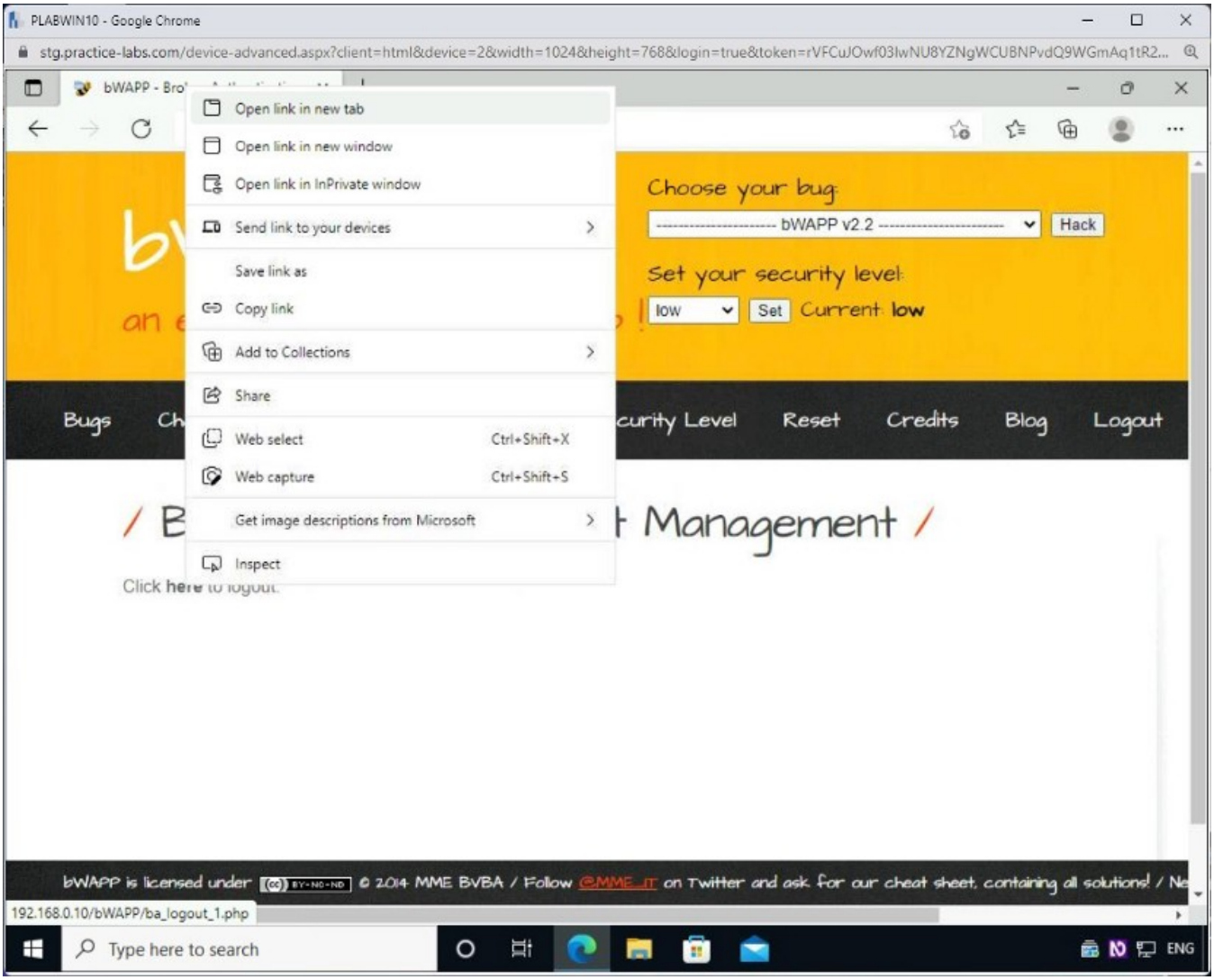
Back on the **bWAPP-Broken Authentication** tab, From the **Choose your bug** drop-down, select **Broken Authentication — Logout Management,** and click **Hack**.



Step 12

The **Broken Auth — Logout Management** webpage is loaded.

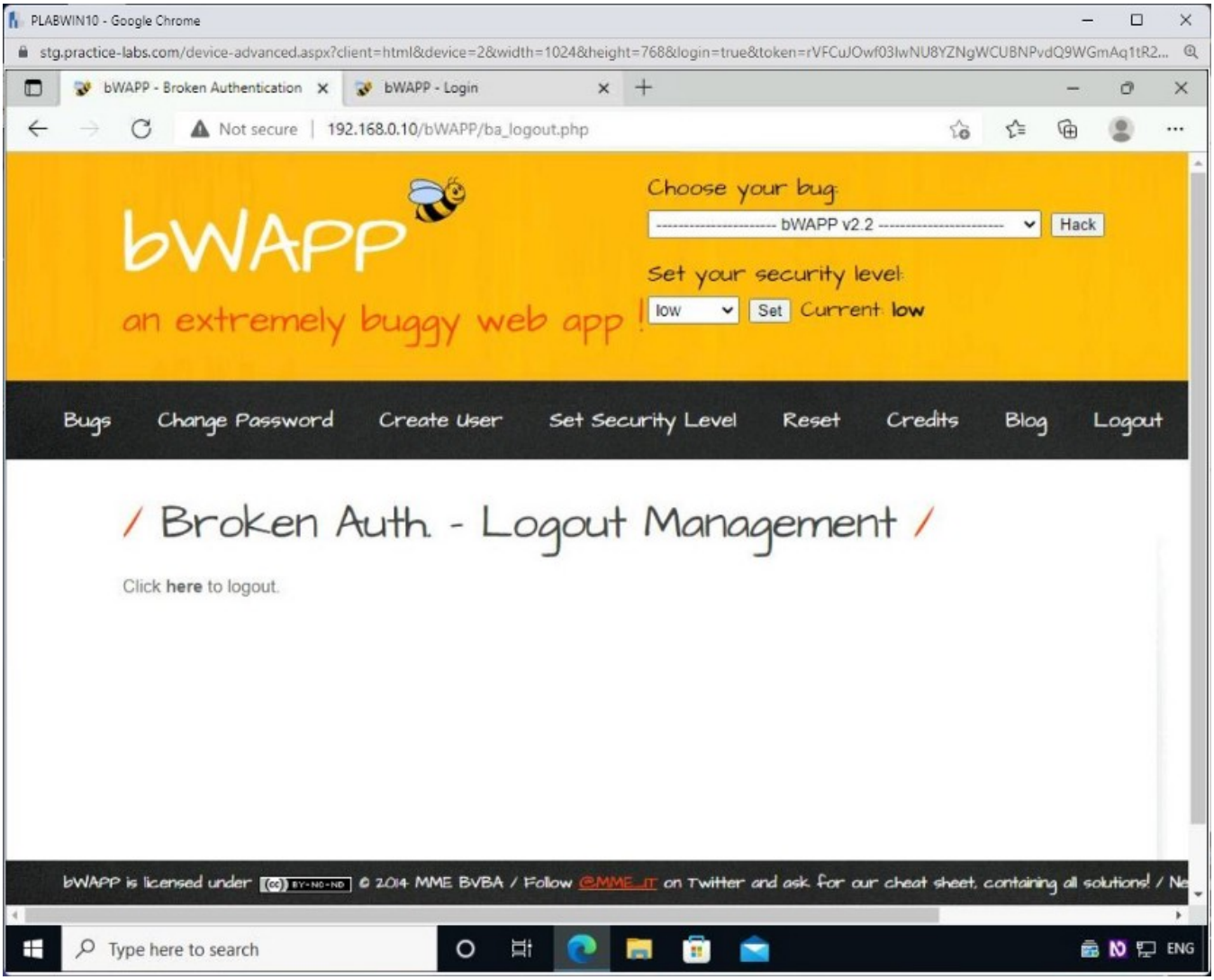
Right-click the **here** in the **Click here to logout** statement and select **Open in new tab**.



Step 13

The **bWAPP — Login** tab opens.

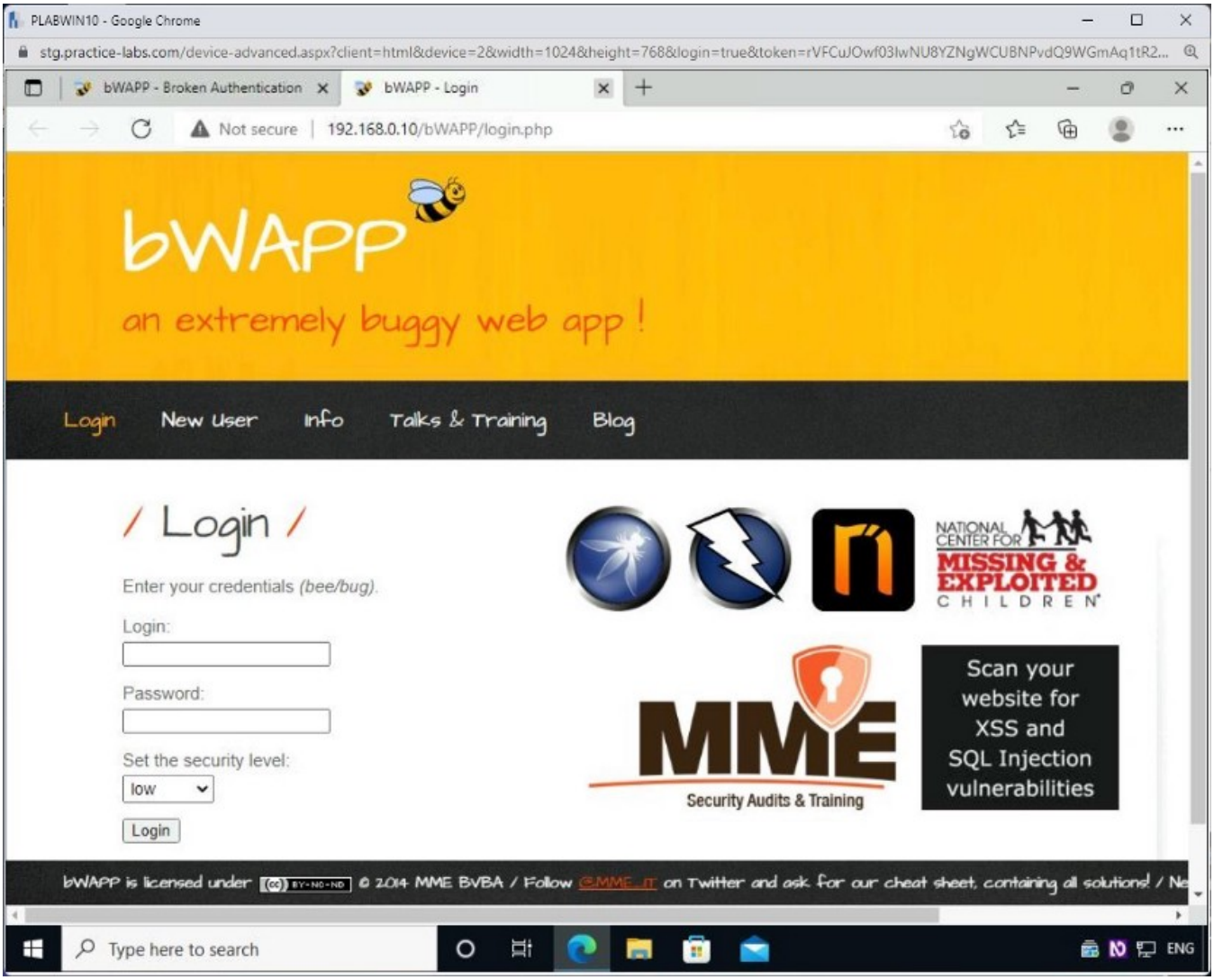
Click the **bWAPP — Login** tab.



Step 14

Notice that you are logged out successfully.

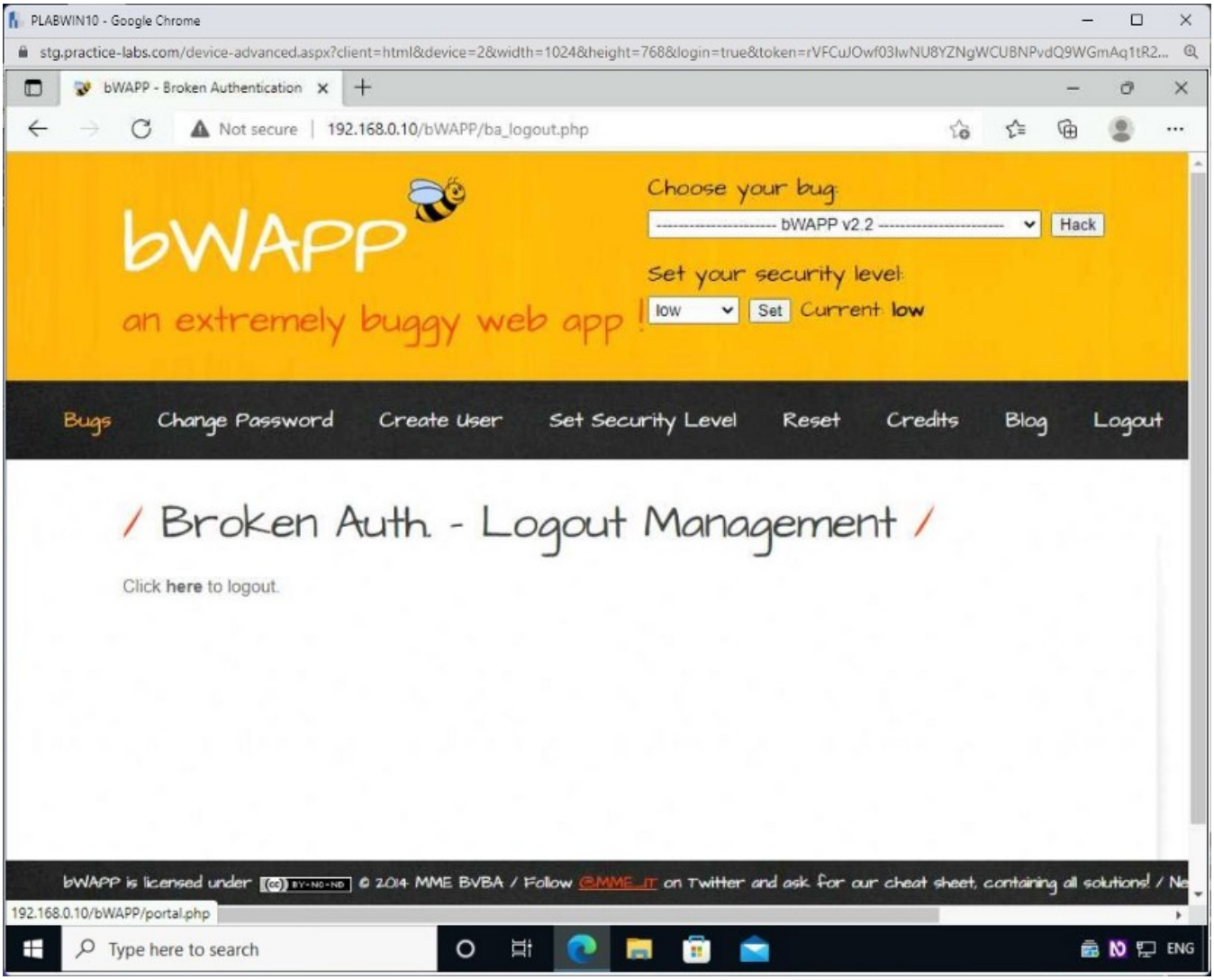
Close this tab.



Step 15

Notice that you are back on the first original tab. You are still logged in to this tab.

Click **Bugs**.



Step 16

The **Portal** webpage is loaded successfully.

This means that you were still able to work within the web application.

Keep the Microsoft Edge window open.

Task 4 — Conduct OS Command Injection Attack

Command injection attacks occur because input fields on an application accept arbitrary data without sufficient input validation. An attacker can use various commands, both for Linux and Windows, to discover required information. For example, an attacker may use the **net use** command to know the existing users in an operating system.

In this task, you will learn to conduct an OS command injection attack. To do this, perform the following step:

Step 1

Connect to **PLABWIN10**. The **Microsoft Edge** window should be open on the beep portal.

From the **Choose your bug** drop-down, select **OS Command Injection,** and click **Hack**.

Step 2

The **OS Command Injection** webpage is displayed. Notice that in the DNS Lookup text box, the following text is entered by default:

```
www.nsa.gov
```

Click **Lookup**.

***Note**: this may take up to a minute to execute.*

Step 3

Notice the output. The firewall blocks the DNS query from going out of the lab environment, and therefore, you get a connection timed out error.

Step 4

Replace the existing text in the **DNS lookup** text box and type the following:

```
|hostname
```

Click **Lookup**.

Step 5

The output is now displayed.

In the notification bar regarding **AutoComplete** to remember web entries, click **No**.

Step 6

Replace the existing text in the **DNS lookup** text box and type the following:

```
|net user
```

Click **Lookup**.

Step 7

The output is now displayed. There are two users, **nobody** and **bee**.

Keep the Microsoft Edge window open.

Task 5 — Perform Server-side Includes Injection Attack (SSI)

SSIs directives are used in web applications to provide dynamic content to HTML pages. SSIs execute a set of defined actions before a webpage is loaded. Therefore, the server which hosts the application, analyses the SSI before the HTML page is loaded in a user's web browser.

The SSI directives are injected in input fields and sent to the webserver. When the field input is submitted, the script, which was added to the input field, is executed. In an SSI attack, an attacker injects malicious scripts in HTML pages.

To perform an SSI attack, perform the following steps:

Step 1

Ensure you have powered on all the devices listed in the introduction and connect to **PLABWIN10**. The **Microsoft Edge** window should be open.

From the **Choose your bug** drop-down, select **Server-side Includes (SSI) Injection,** and click **Hack**.

Step 2

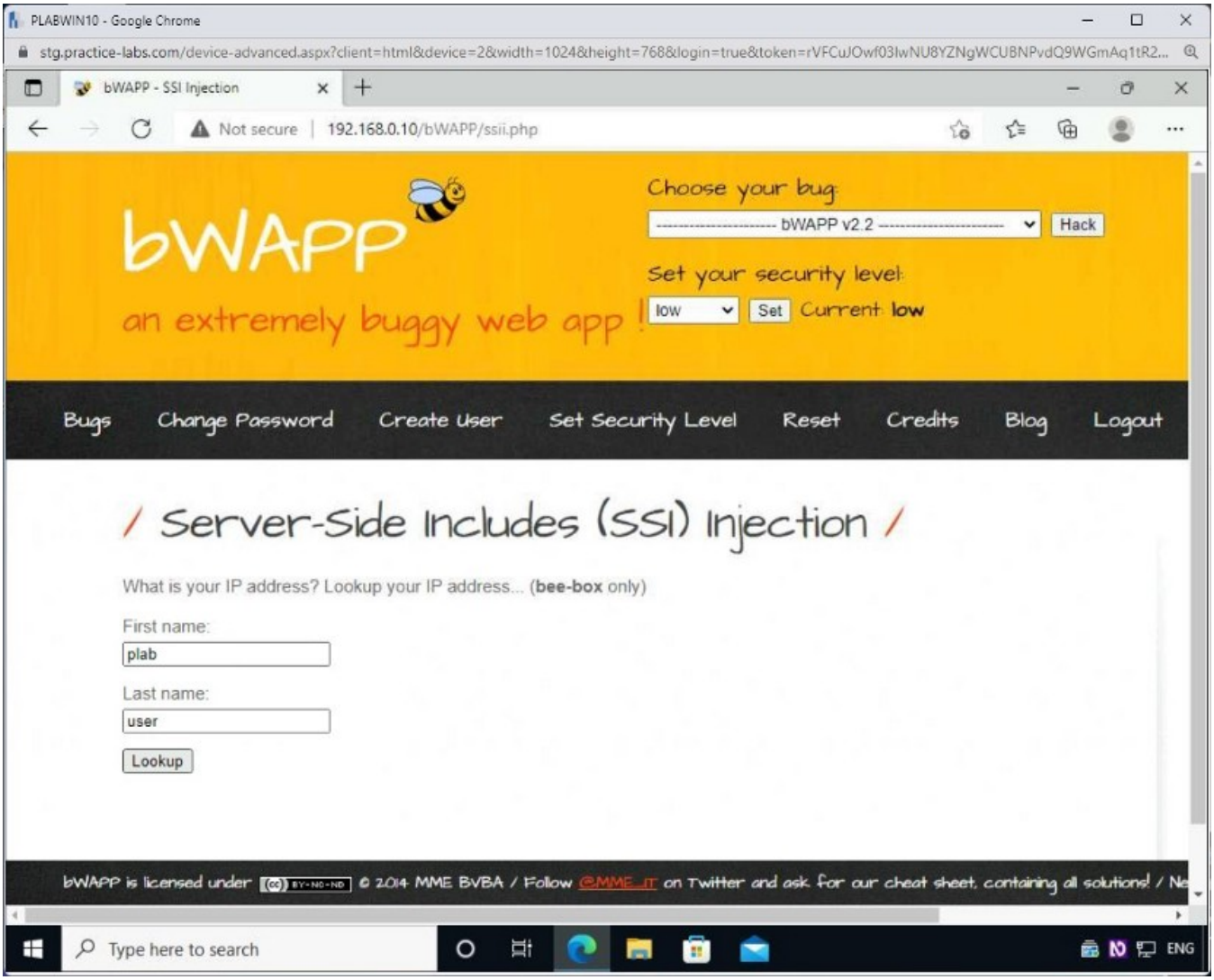
The **Server-Side Includes (SSI) Injection** webpage is loaded. In the **First name** text box, type the following:

```
plab
```

In the **Last name** text box, type the following:

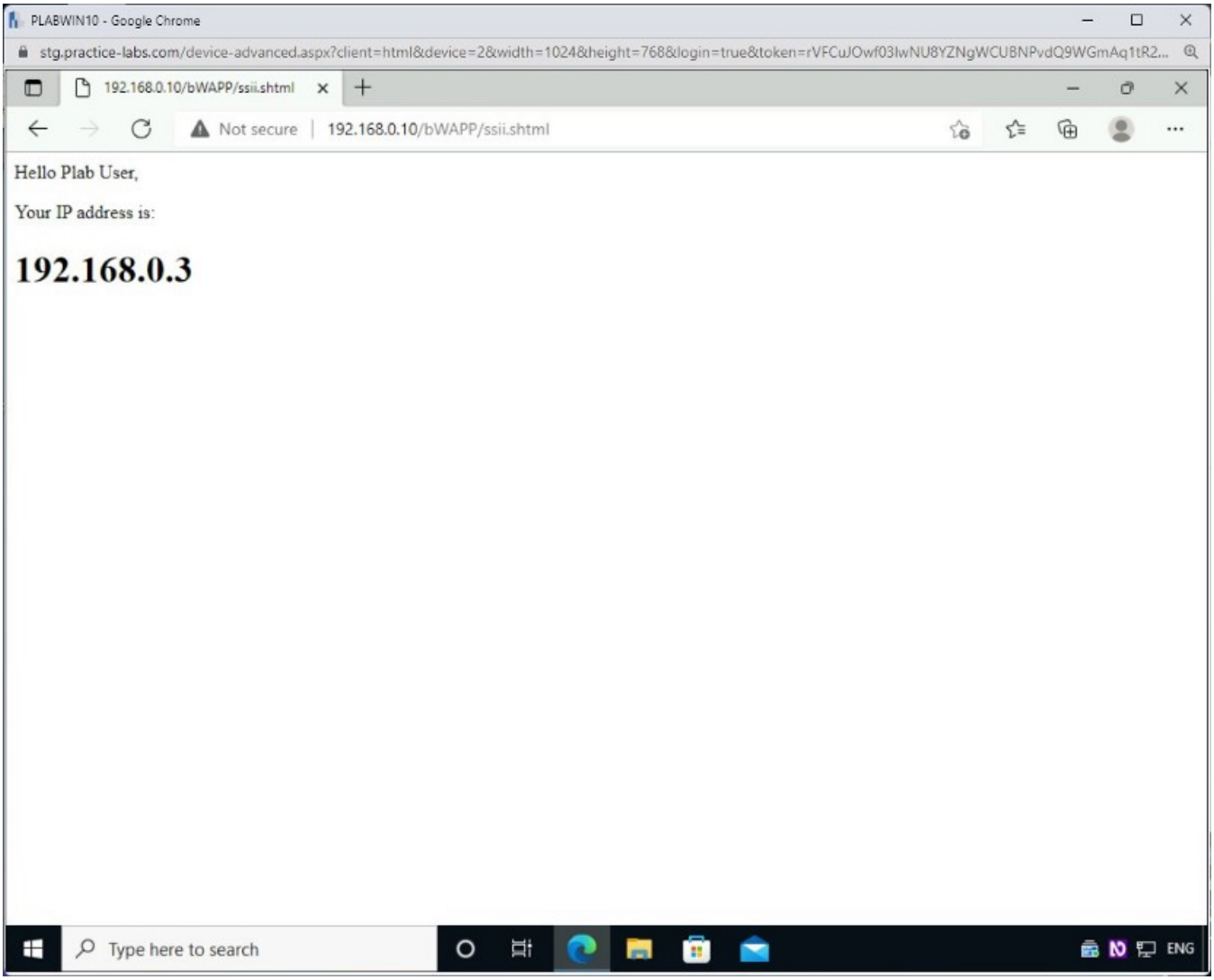
```
user
```

Click **Lookup**.



Step 3

Notice that the output is displayed with an IPv4 address.



Step 4

Click the **Back** arrow to navigate back to the previous page.

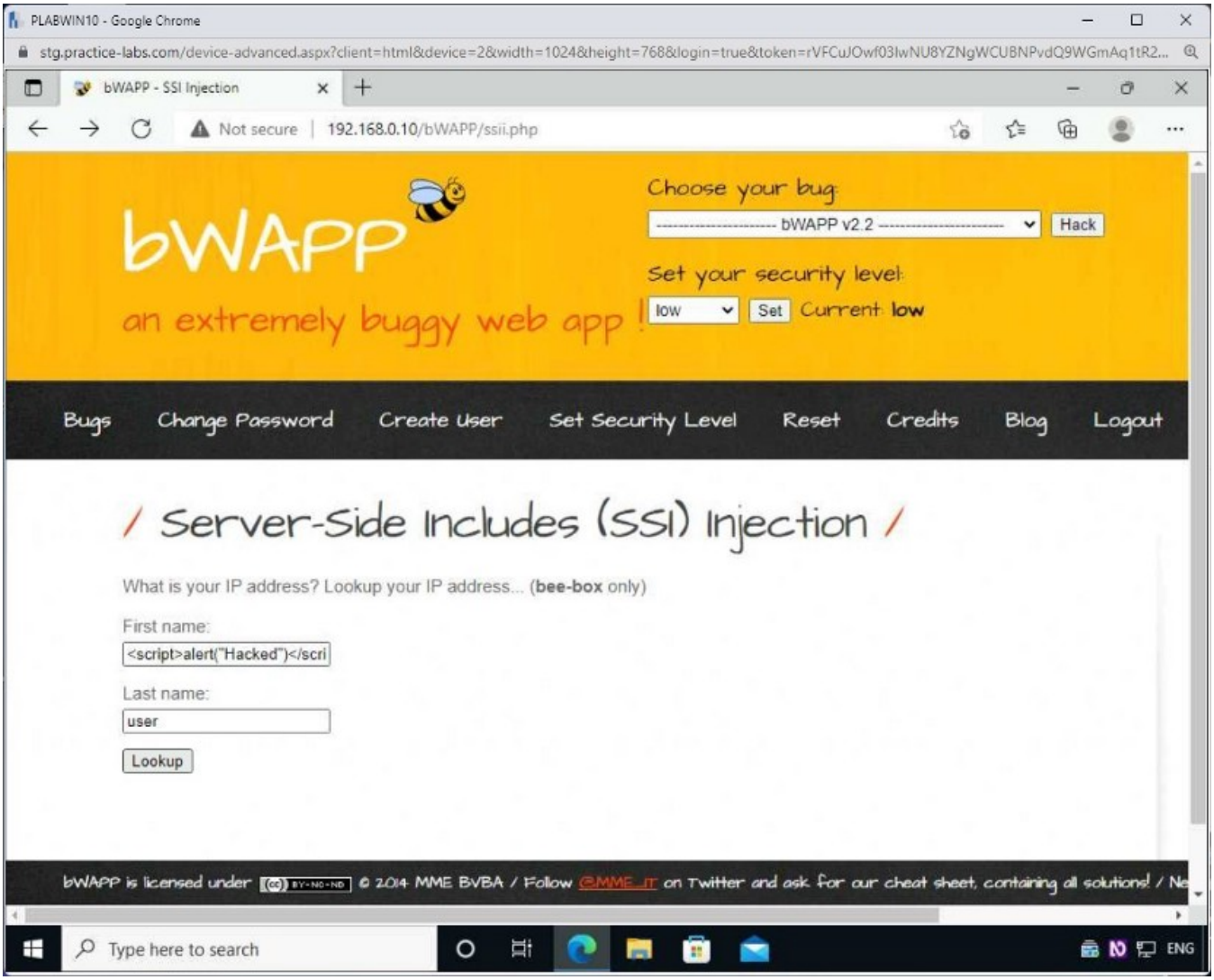
Back on the **Server-Side Includes (SSI) Injection** webpage, you need to insert the following into the **First name** text box:

```
<script>alert("Hacked")</script>
```

In the **Last name** text box, type the following:

```
user
```

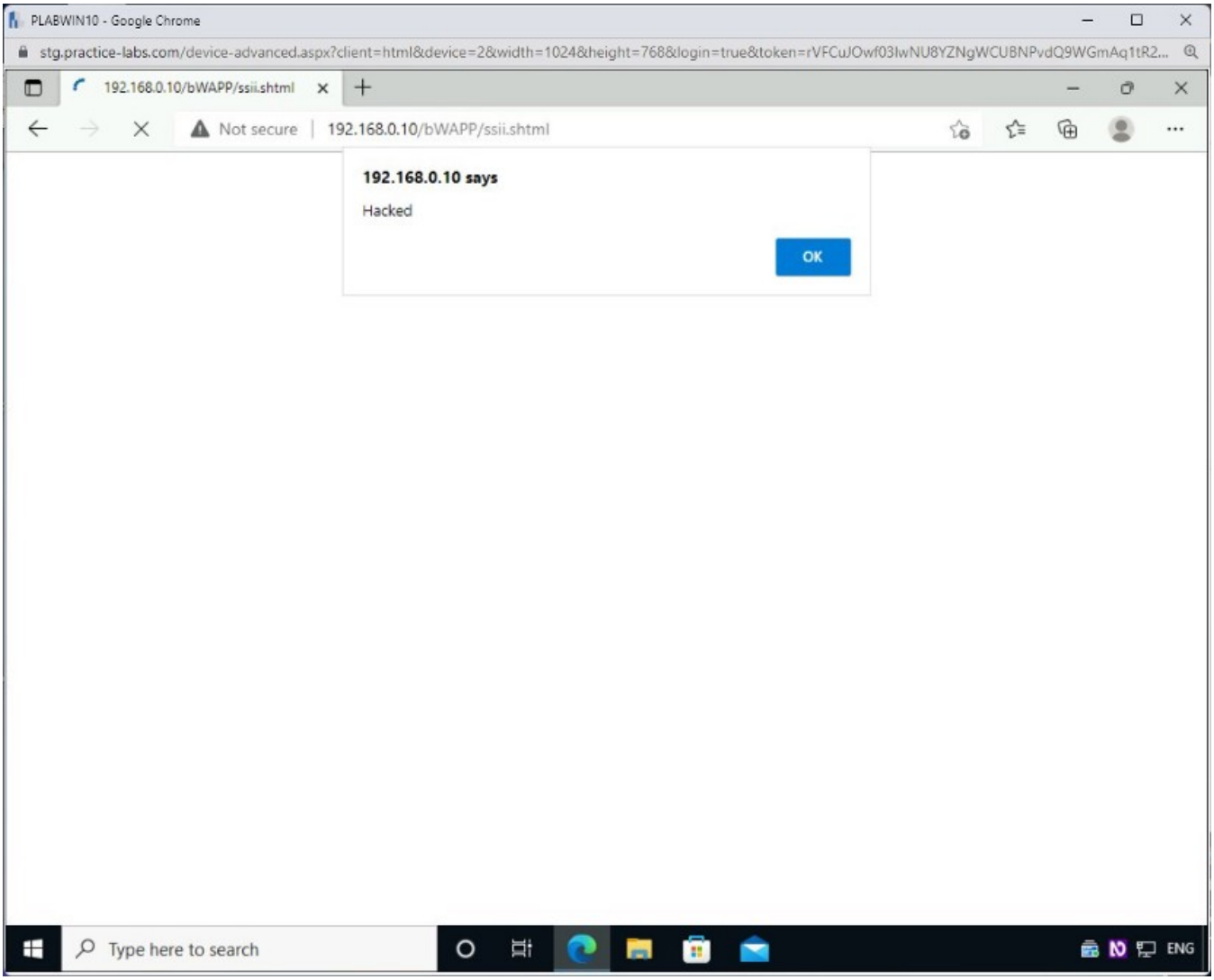
Click **Lookup**.



Step 5

Notice the output. The script embedded in the **First name** text box has been executed successfully.

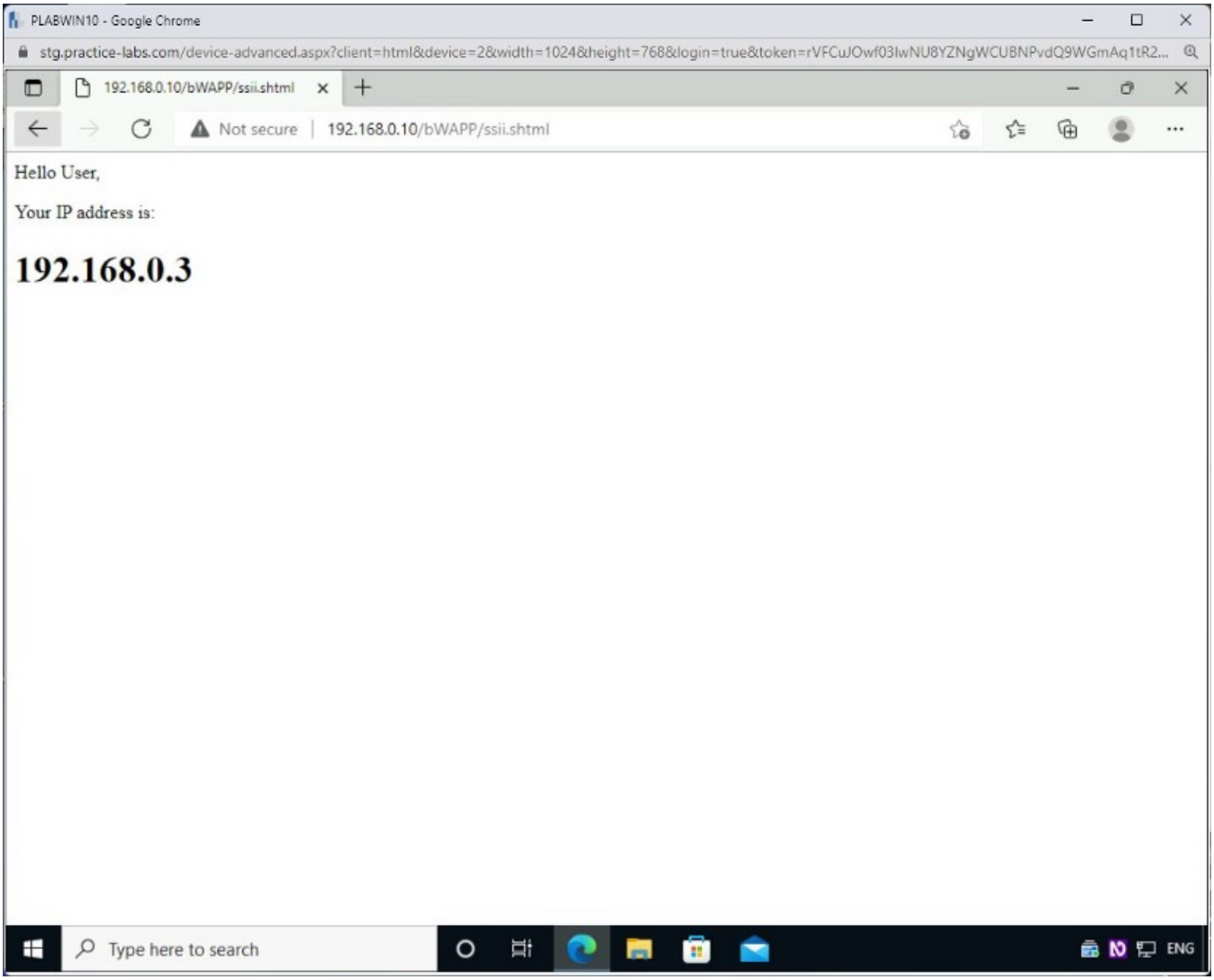
Click **OK**.



Step 6

Again, the text on the webpage is changed to the IPv4 address.

Click the **Back** button on the browser.



Step 7

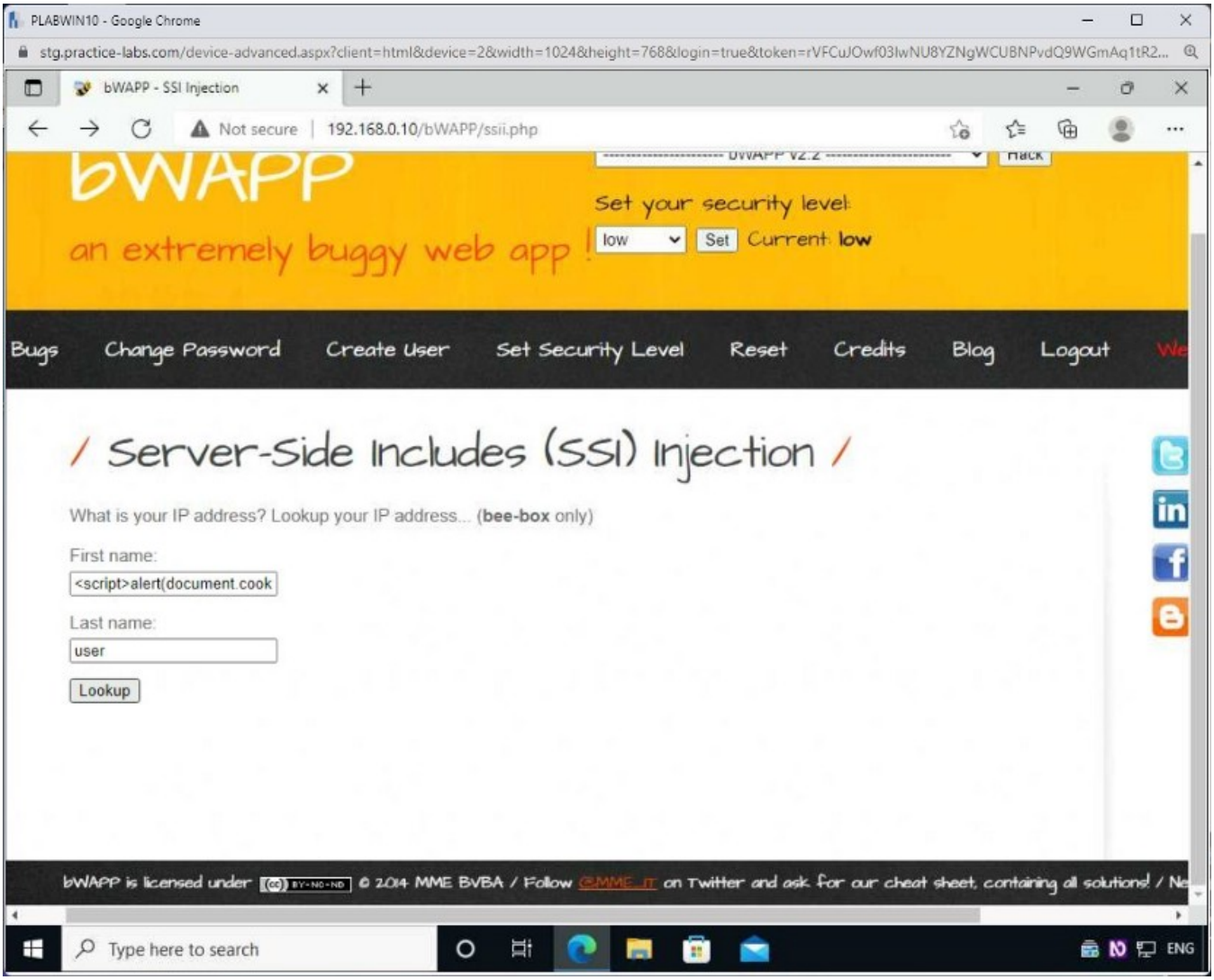
Back on the **Server-Side Includes (SSI) Injection** webpage, you can also fetch the cookie from the web server. you need to insert the following into the **First name** text box:

```
<script>alert(document.cookie)</script>
```

In the **Last name** text box, type the following:

```
user
```

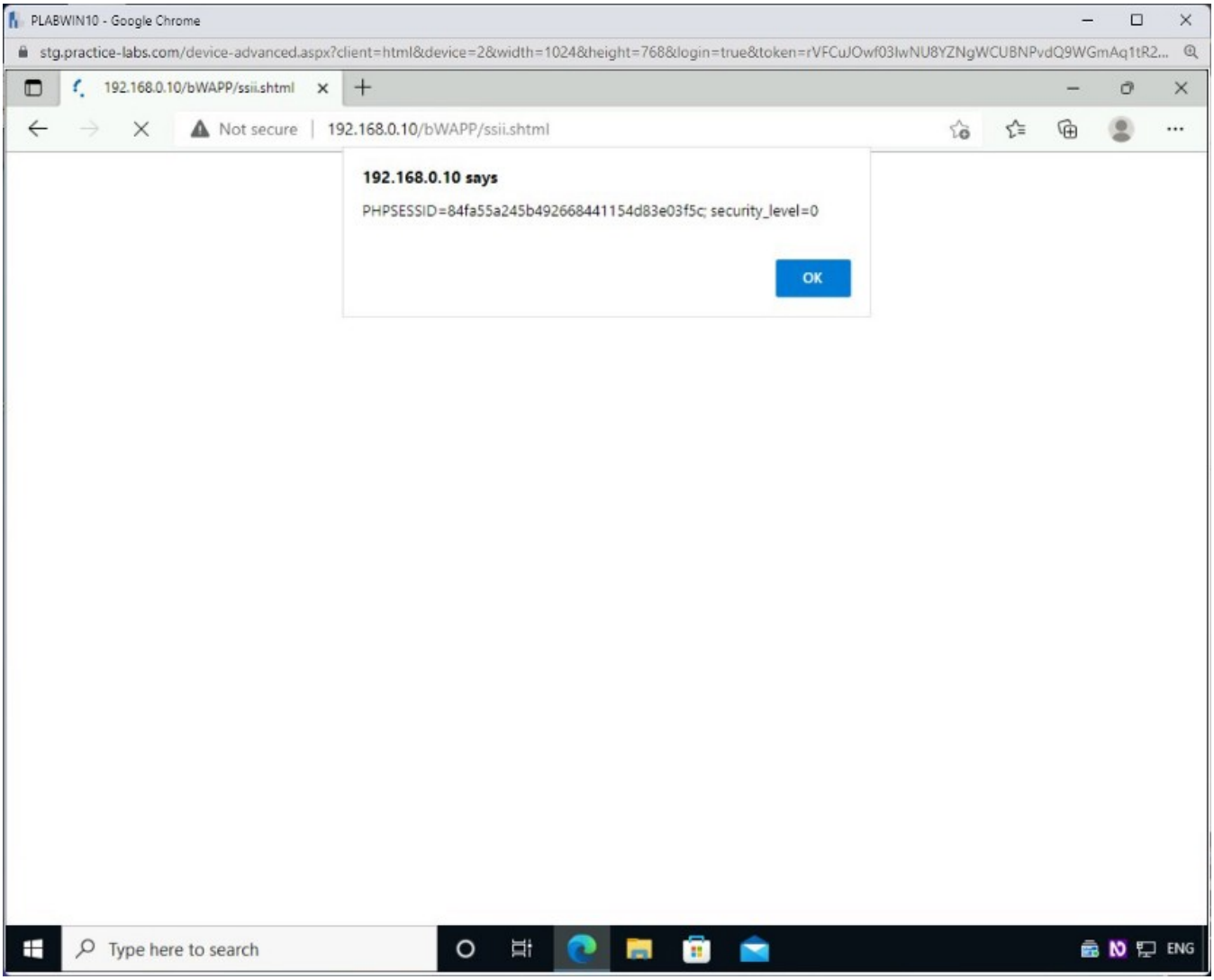
Click **Lookup**.



Step 8

Notice the output. The script embedded in the **First name** text box has been executed successfully.

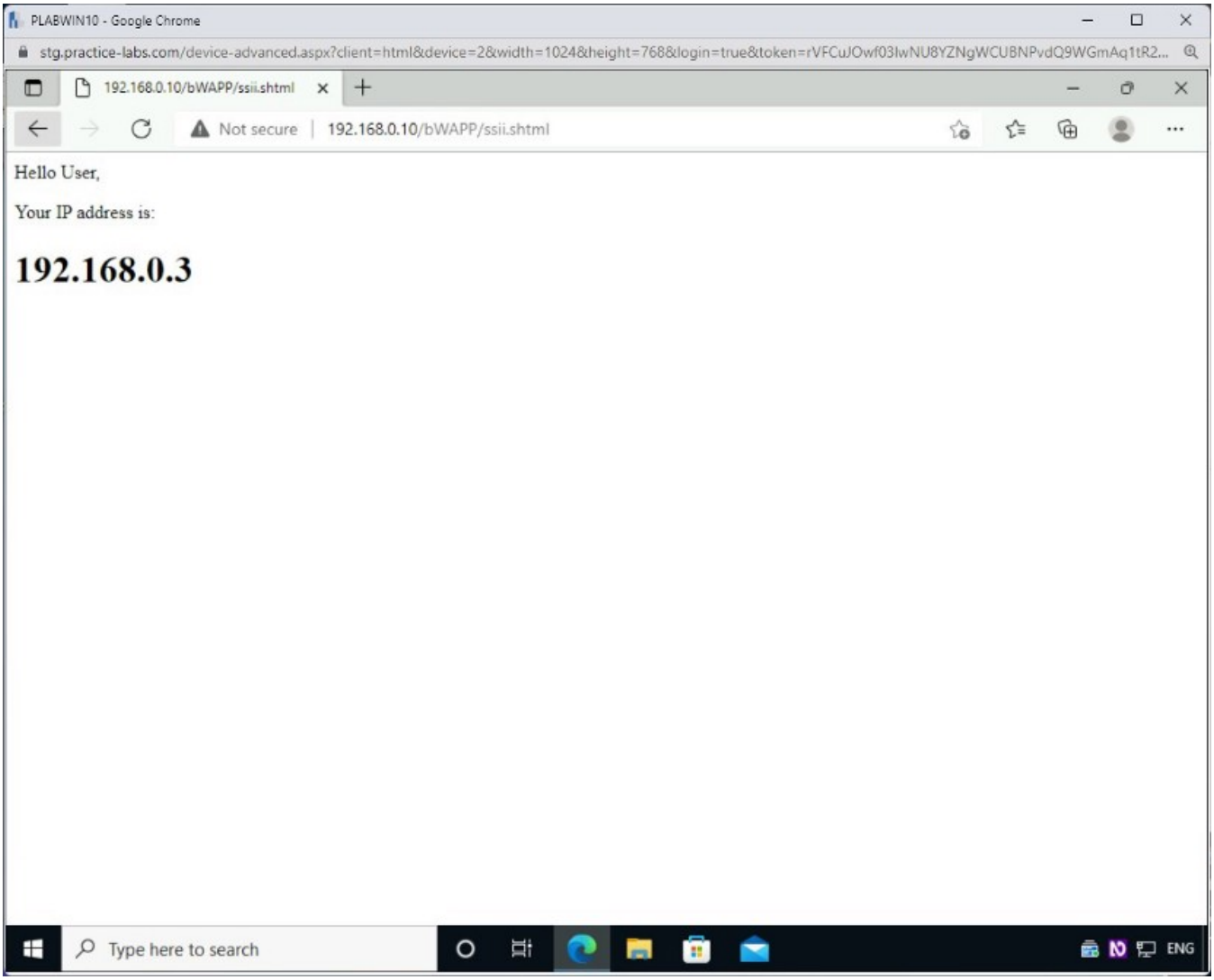
The cookie is being displayed in the dialog box. Click **OK**.



Step 9

Again, the text on the webpage is changed to the IPv4 address.

Click the **Back** button on the browser.



Step 10

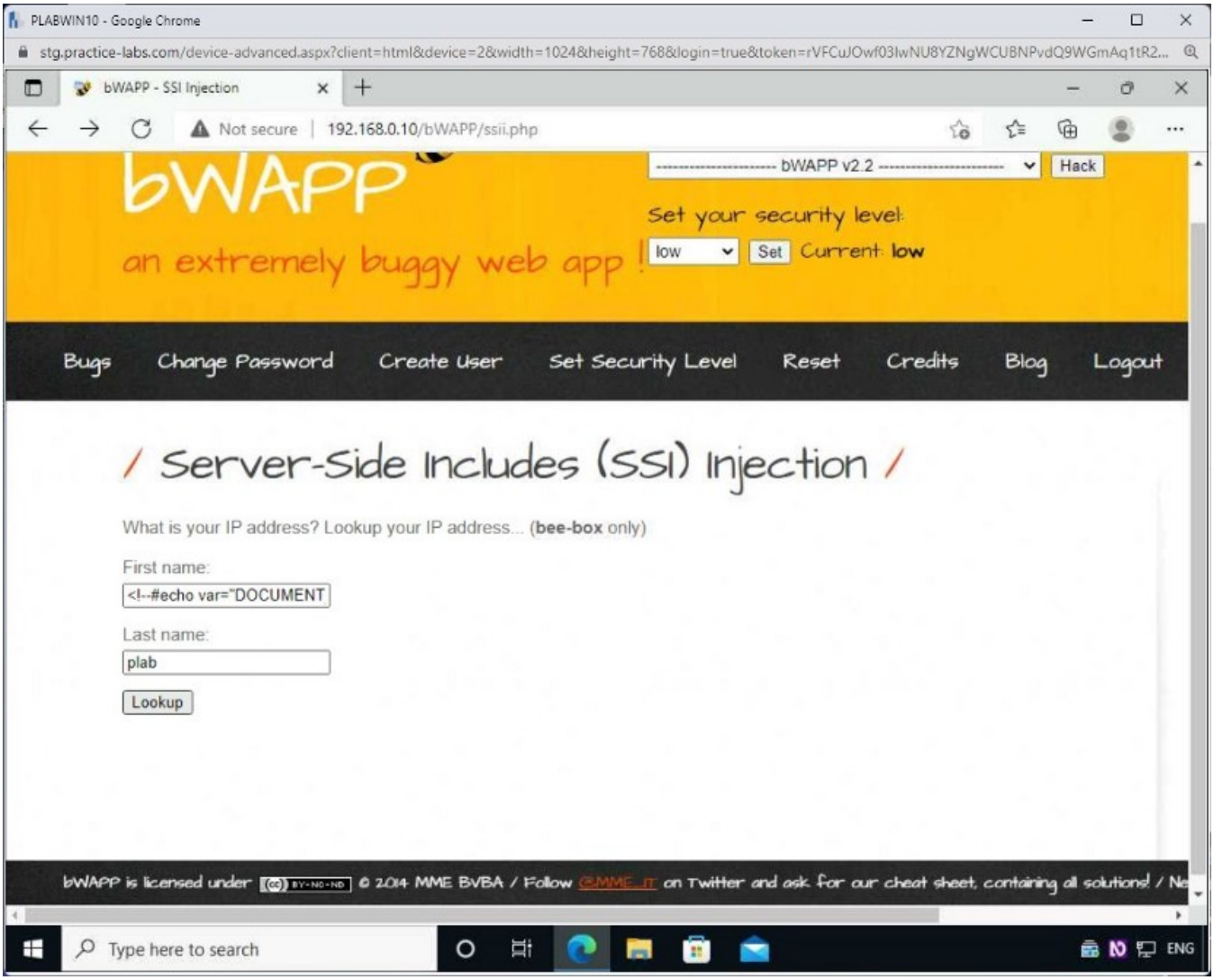
Back on the **Server-Side Includes (SSI) Injection** webpage, you can show the current document name. you need to insert the following into the **First name** text box:

```
<!--#echo var="DOCUMENT_NAME" -->
```

In the **Last name** text box, type the following:

```
plab
```

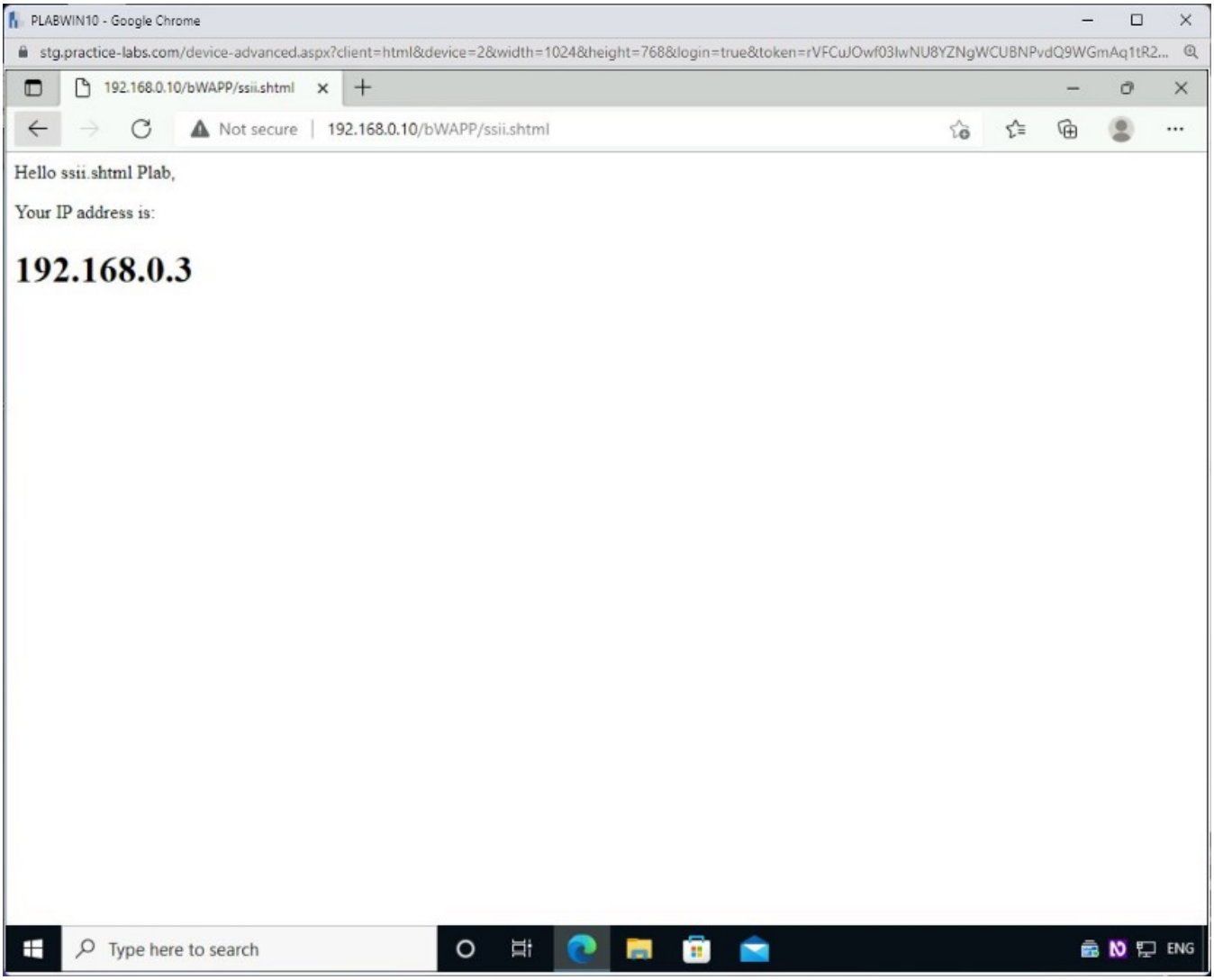
Click **Lookup**.



Step 11

Notice the output. It displays the current document name before the last name, plab.

Click the **Back** button on the web browser window.



Step 12

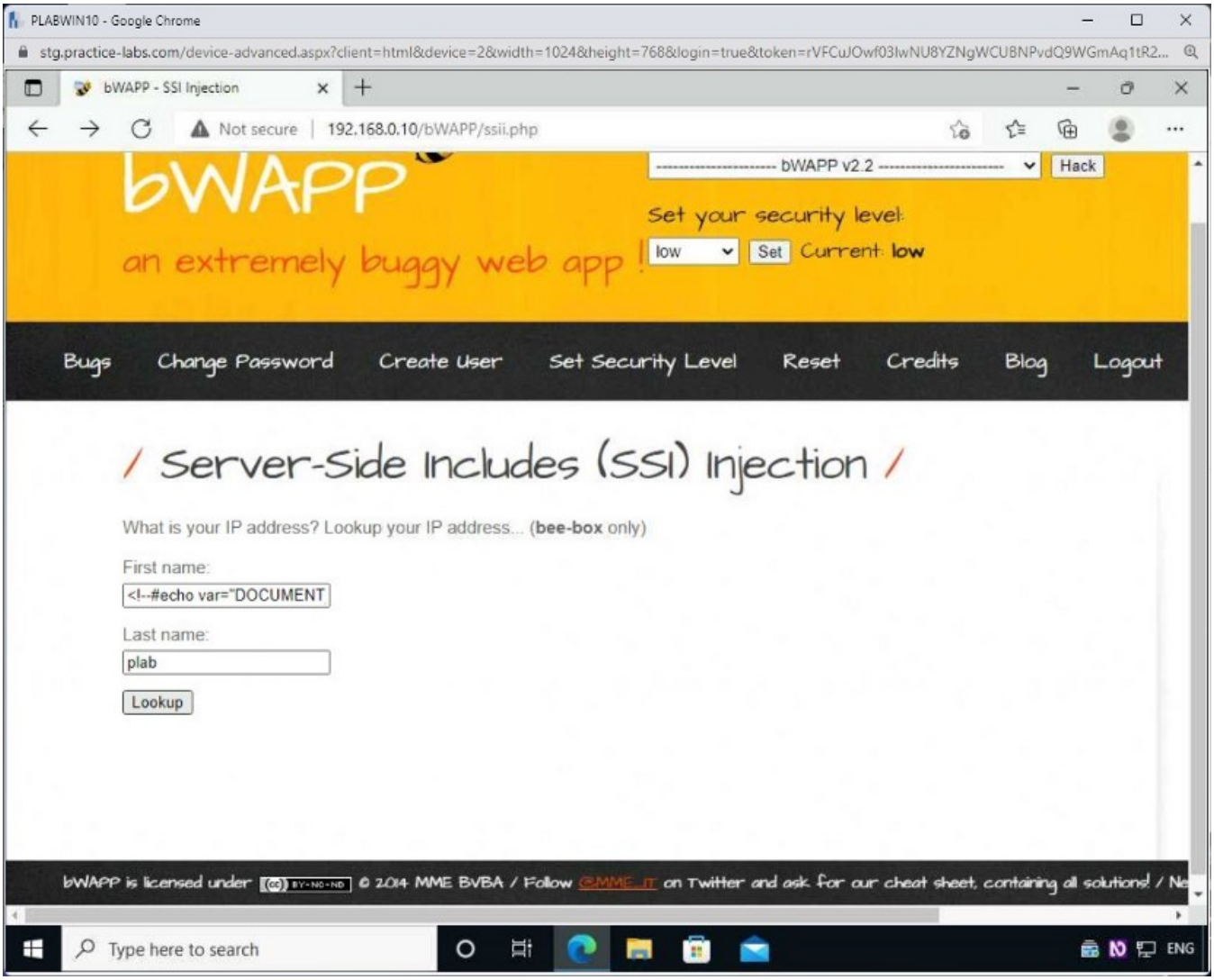
Back on the **Server-Side Includes (SSI) Injection** webpage, you can show the virtual path and filename. To do this, you need to insert the following into the **First name** text box:

```
<!--#echo var="DOCUMENT_URI" -->
```

In the **Last name** text box, type the following:

```
plab
```

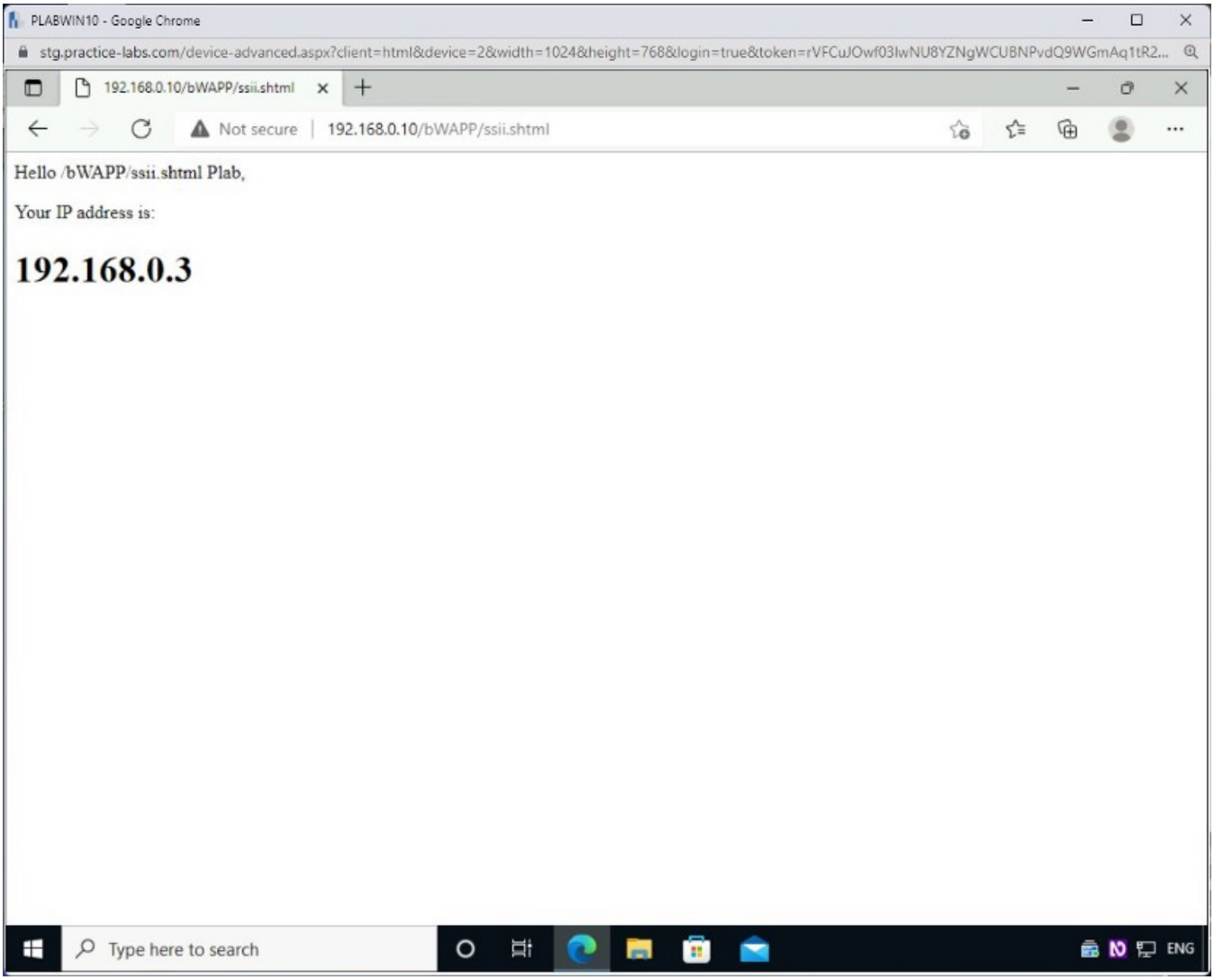
Click **Lookup**.



Step 13

Notice that the output shows the virtual path and the file name.

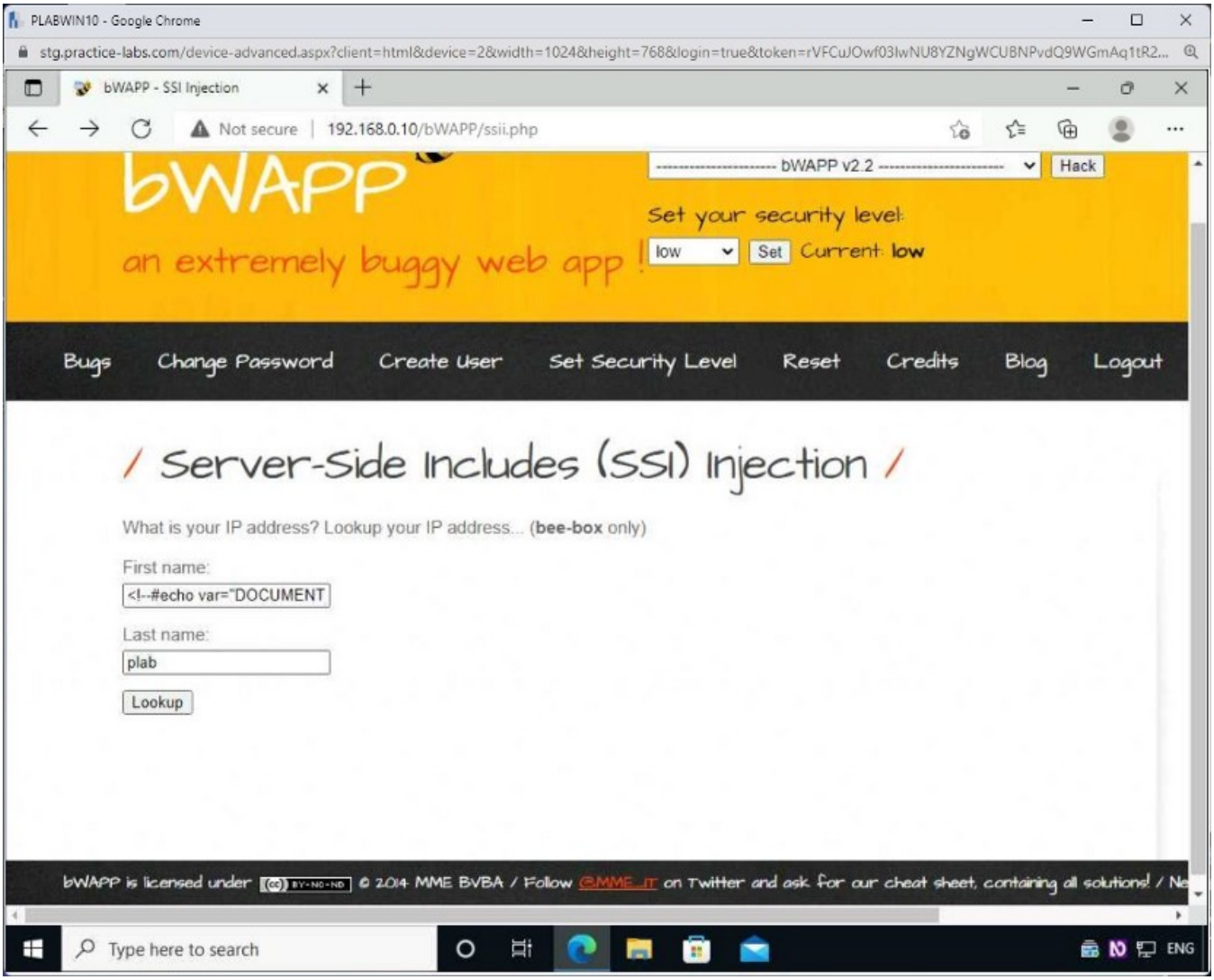
Click the **Back** button on the web browser.



Step 14

Click the **Back** button on the web browser window.

You should be back on the **Server-Side Includes (SSI) Injection** webpage.



Keep the web browser window open.

Task 6 — Perform Cross-site Scripting Attack

Cross-site scripting (XSS) is an attack where an attacker injects malicious scripts into vulnerable web applications, thereby causing serious damage. The malicious script is executed in a victim's web browser when they visit the infected page. JavaScript is the most common scripting language used for developing malicious code. There are two broad categories of cross-site scripting attacks. These are as follows:

- Reflected XSS
- Stored XSS

A reflected XSS attack is also known as a non-persistent XSS attack. An attacker sends the malicious code as a link in an email or posts it on a website in this type of attack. Once a victim is infected, the malicious script embedded gets executed and is reflected on a victim's web browser. The browser then sends cookie information of a victim's session to an attacker.

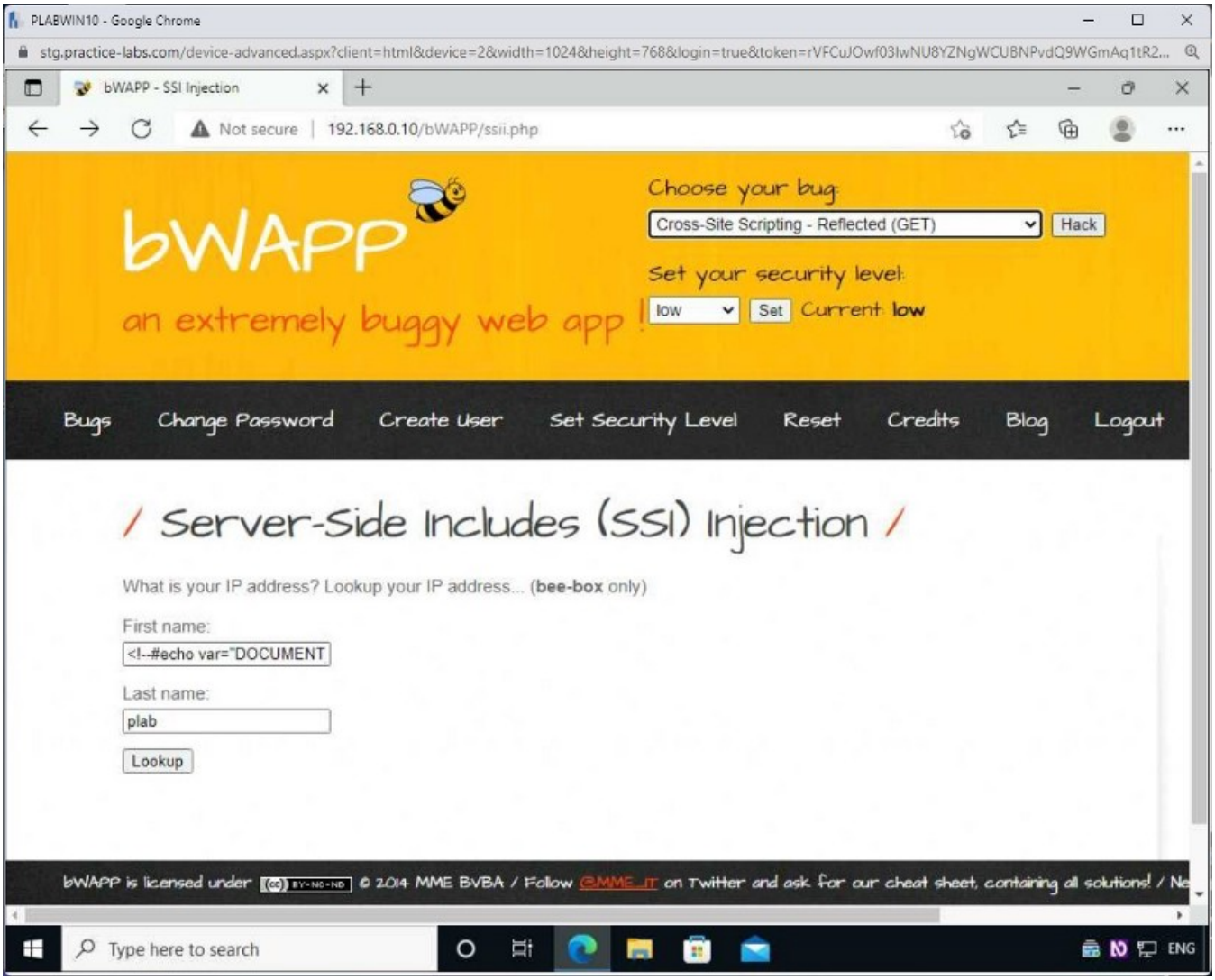
A stored XSS attack is also known as a persistent XSS attack. In this type of attack, an attacker injects malicious script directly into a vulnerable web application. The malicious script steals cookie information of victim's sessions and sends it to an attacker. Therefore, in stored XSS attacks, the risk is higher as malicious script executes on every visit to the application.

In this task, you will perform cross-site scripting attacks on the **bWAPP** application.

Step 1

Connect to **PLABWIN10**. The **Microsoft Edge** window should be open.

To begin reflected cross-site scripting attack, from the **Choose your bug** drop-down, click **Cross-site Scripting — Reflected (GET),** and click **Hack**.



Step 2

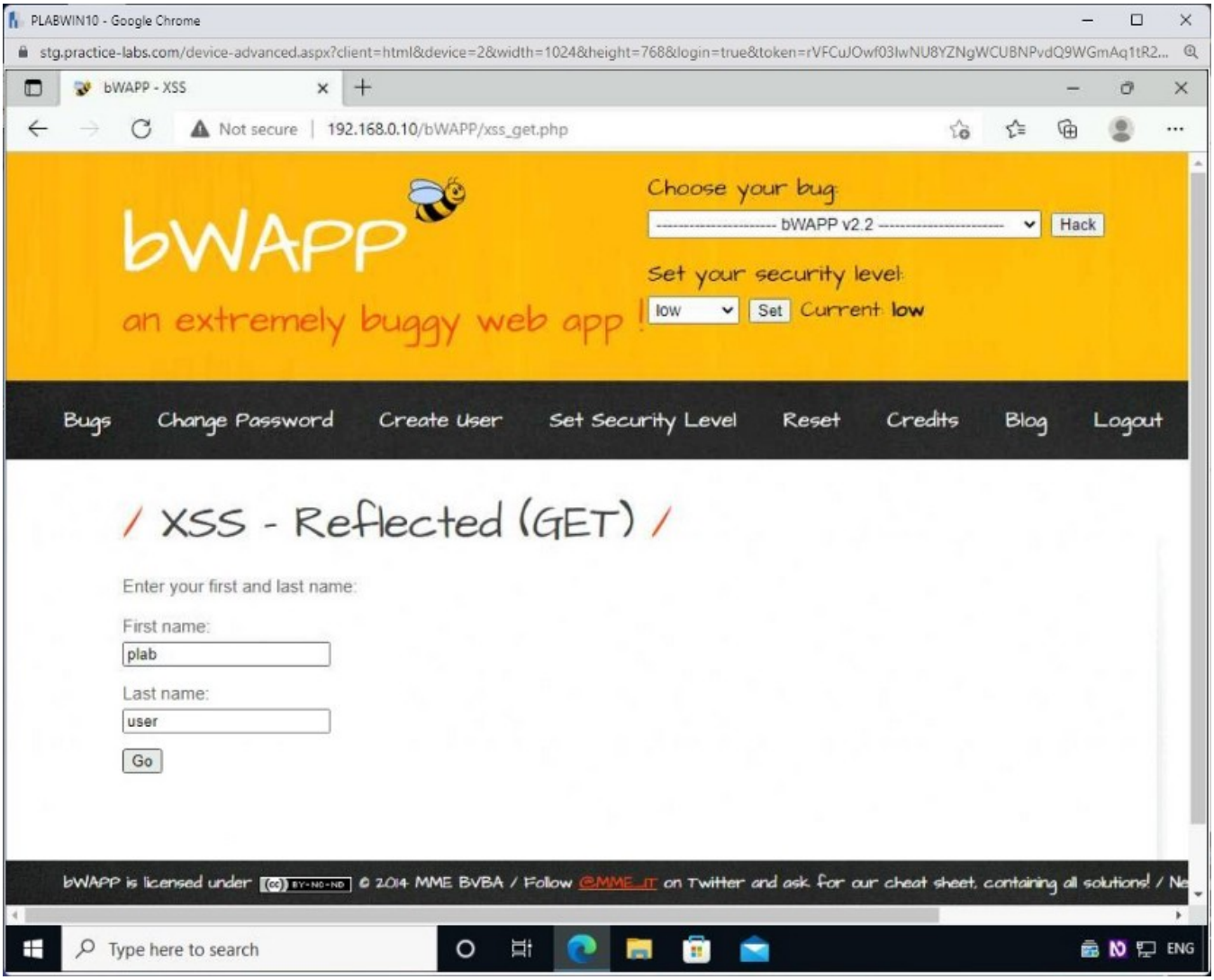
On the **XSS — Reflected (GET)** webpage, In the **First name** text box, type the following name:

```
Plab
```

In the **Last name** text box, type the following name:

```
User
```

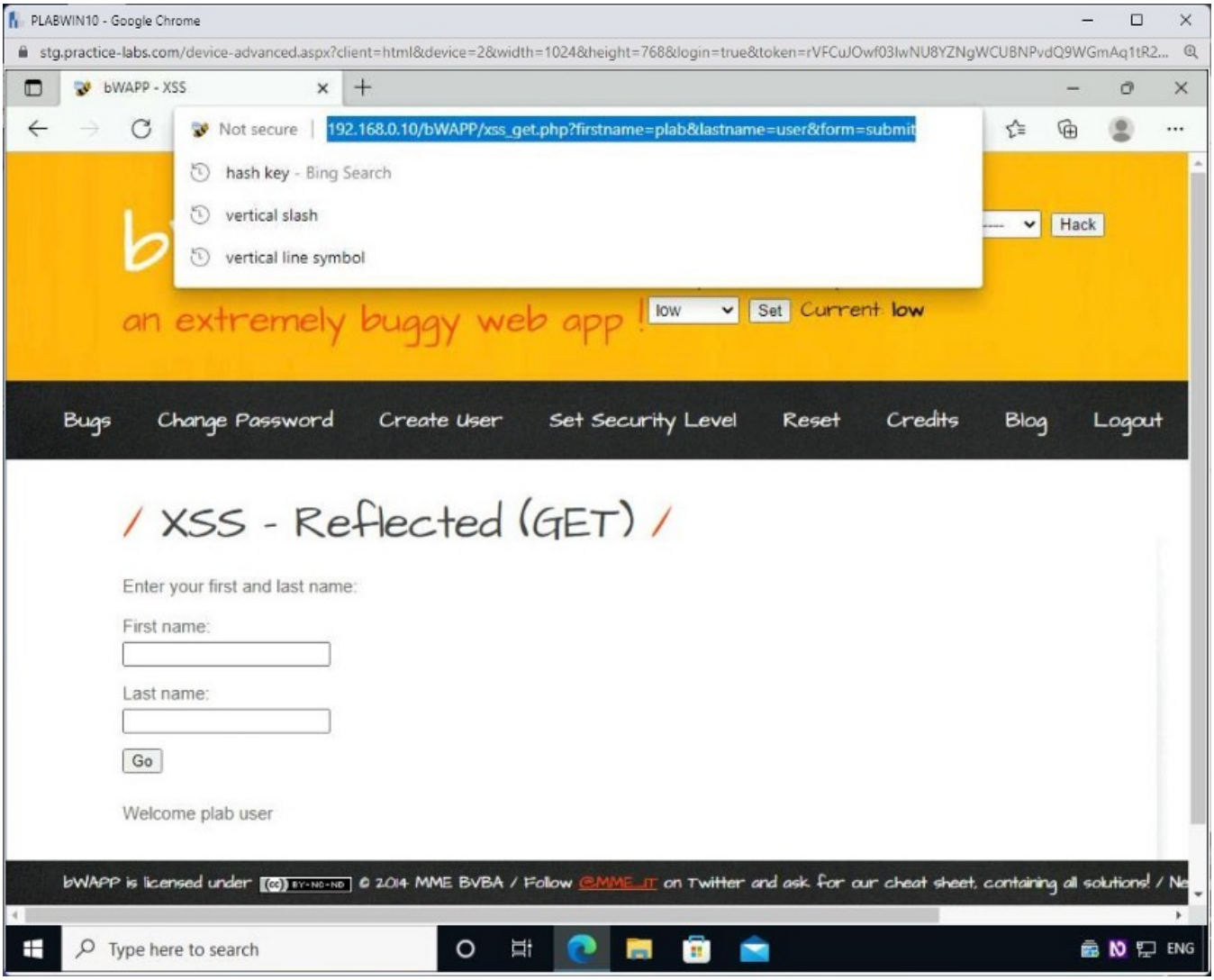
Click **Go**.



Step 3

Observe the output displayed in the address bar.

The input passed to the server is reflected in the application. This indicates a good entry point for reflected XSS attacks as the response is getting reflected.



Step 4

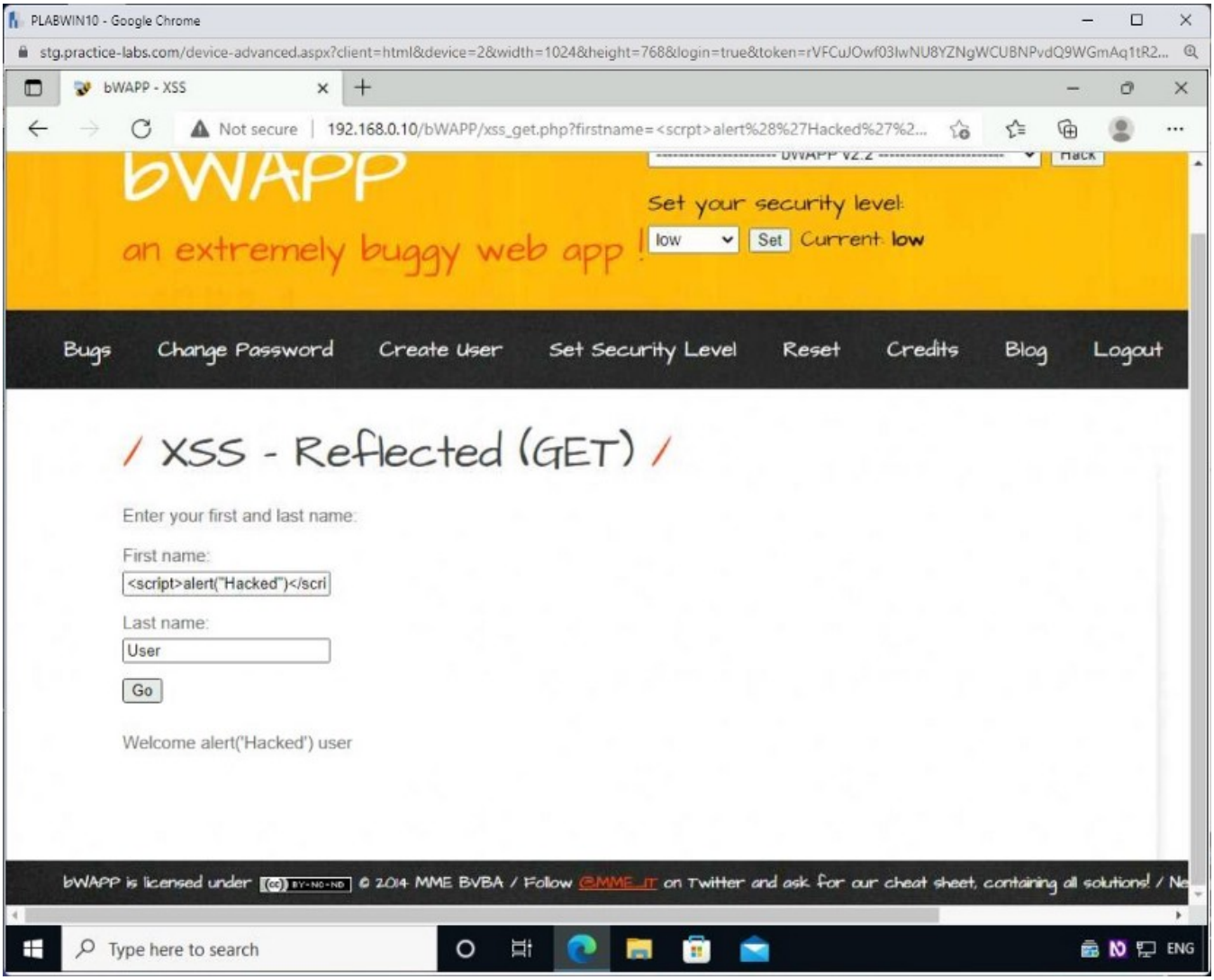
On the **XSS — Reflected (GET)** webpage, In the **First name** text box, inject a JavaScript code by typing the following:

```
<script>alert("Hacked")</script>
```

In the **Last name** text box, type the following name:

```
User
```

Click **Go**.

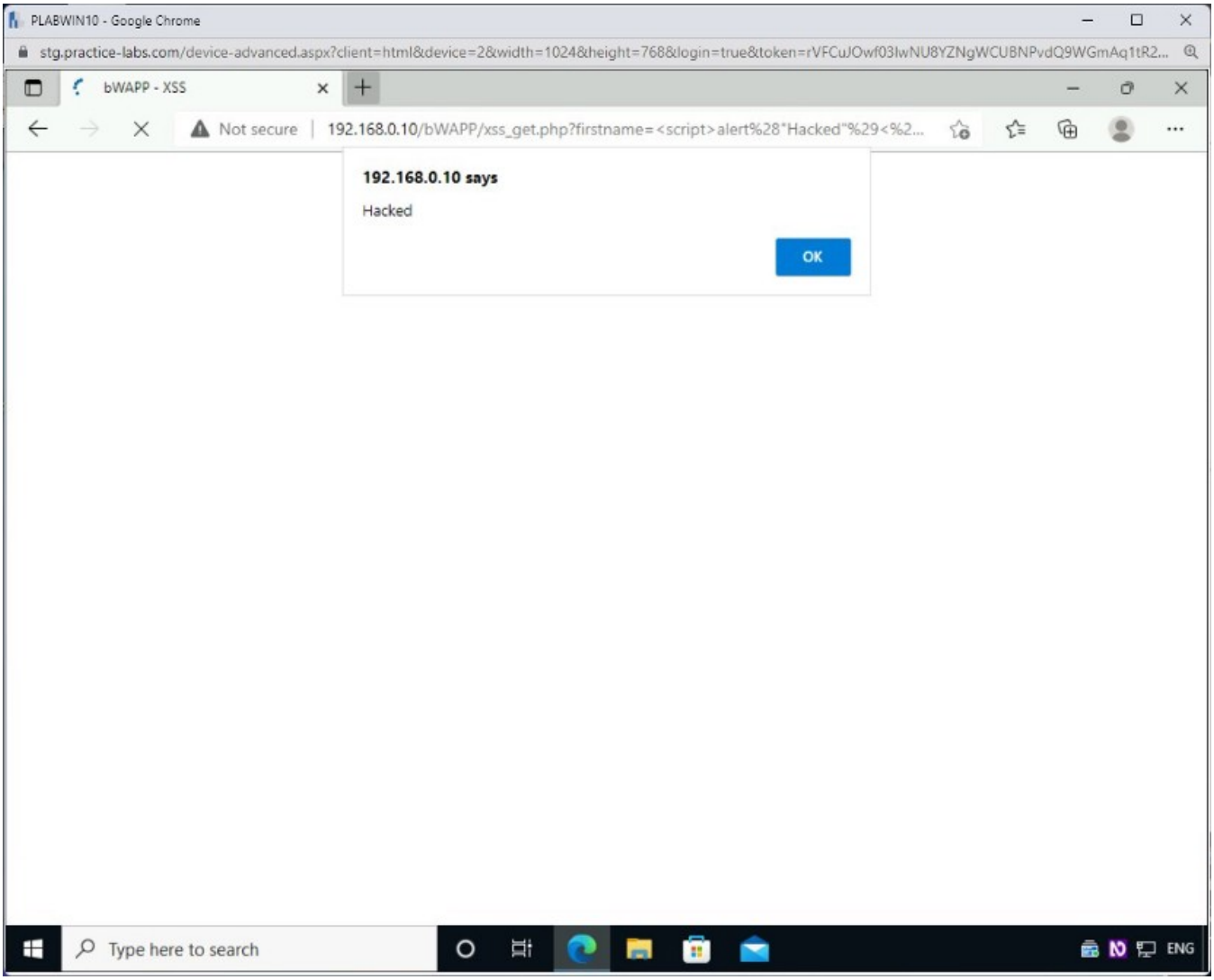


Step 5

The server processes the script and displays the alert message box.

Click **OK**.

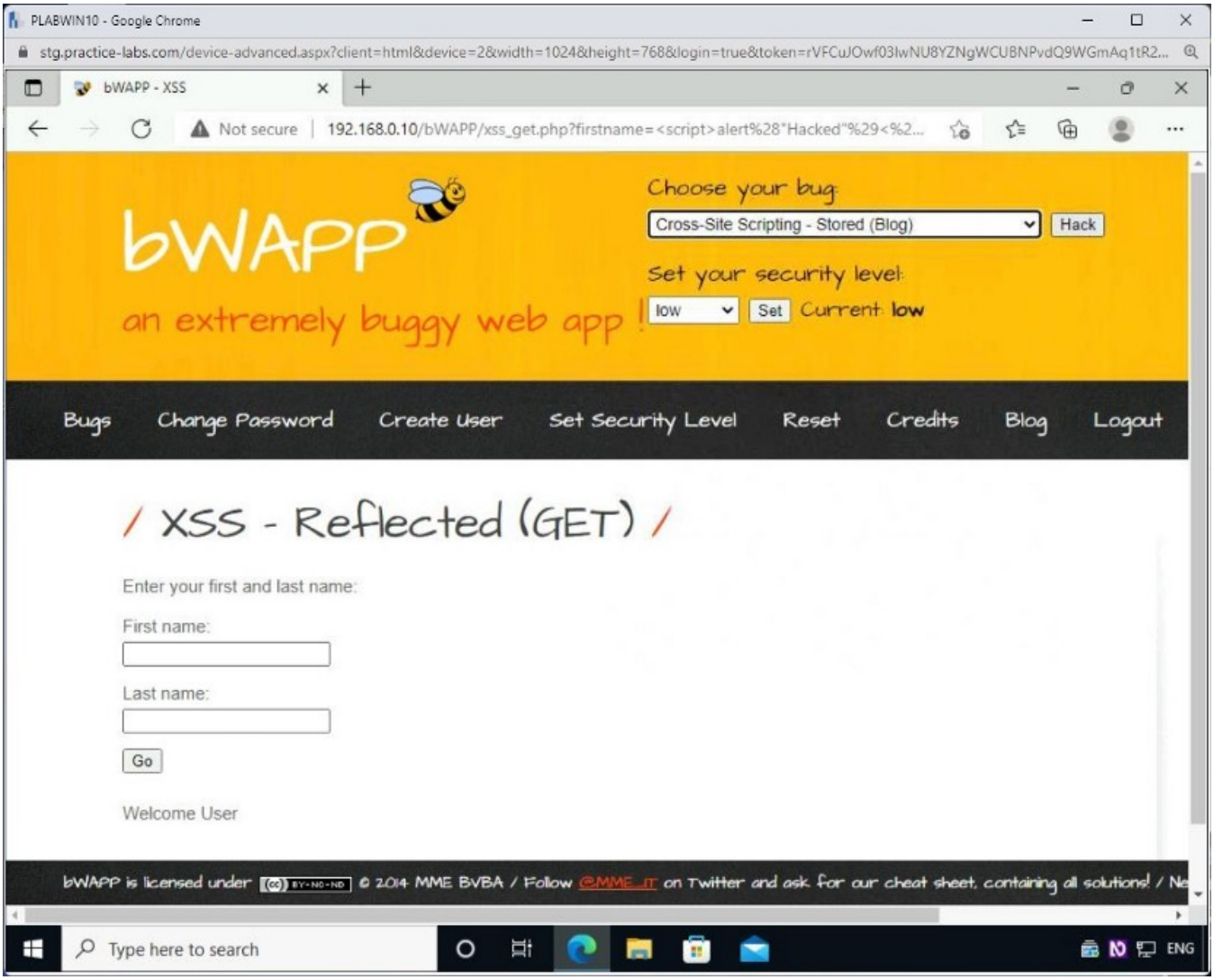
Note: *The script is executed and reflected in the server response, a vulnerability. Using this vulnerability in a real-time environment, an attacker can inject malicious scripts in less secure applications to steal cookies, learn about the document location, and so on.*



Step 6

You are back on the **XSS — Reflected (GET)** webpage.

From the **Choose your bug** drop-down, select **Cross-Site Scripting — Stored (Blog),** and click **Hack**.

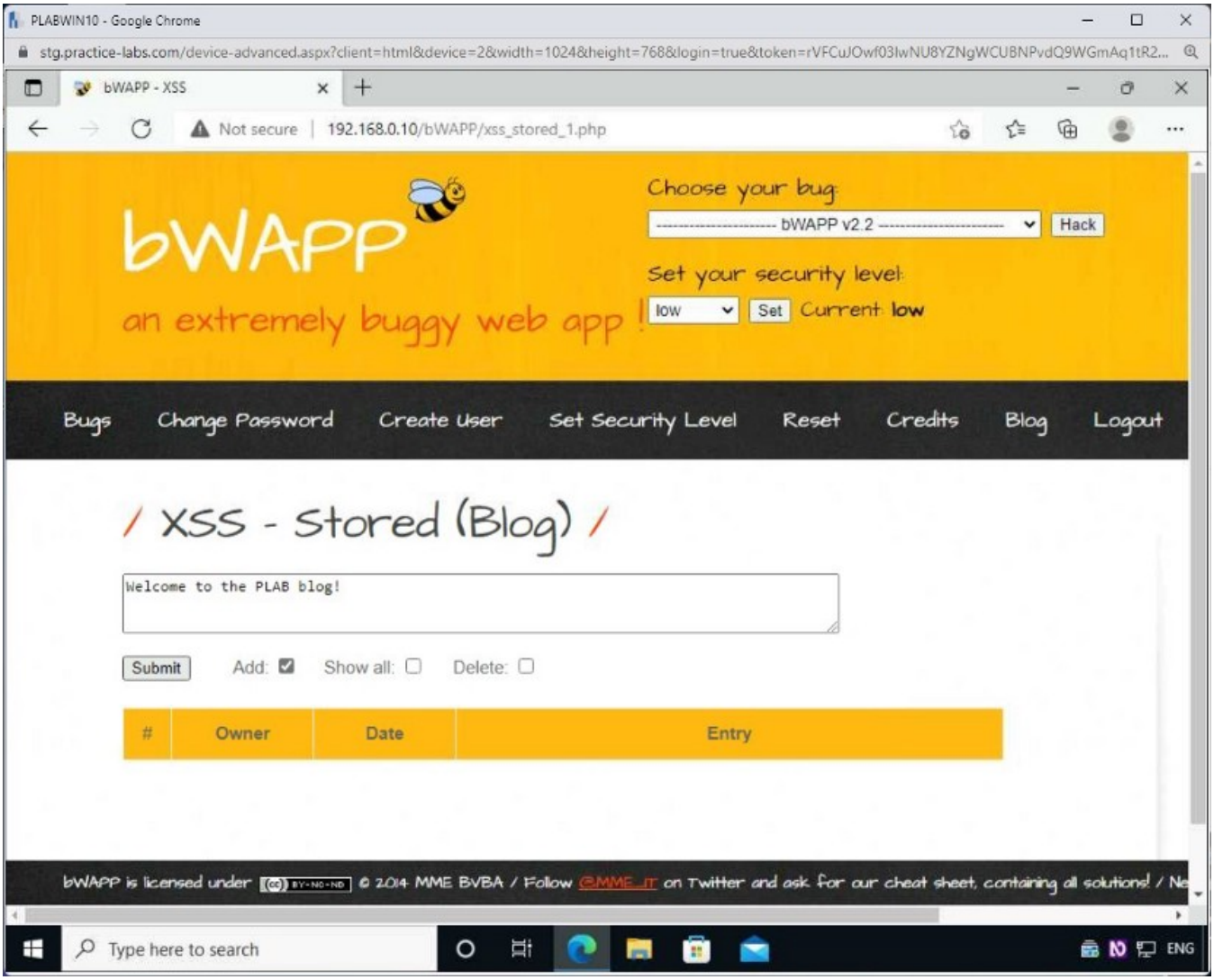


Step 7

On the **XSS — Stored (Blog)** webpage, type the following message in the text box:

```
Welcome to the PLAB blog!
```

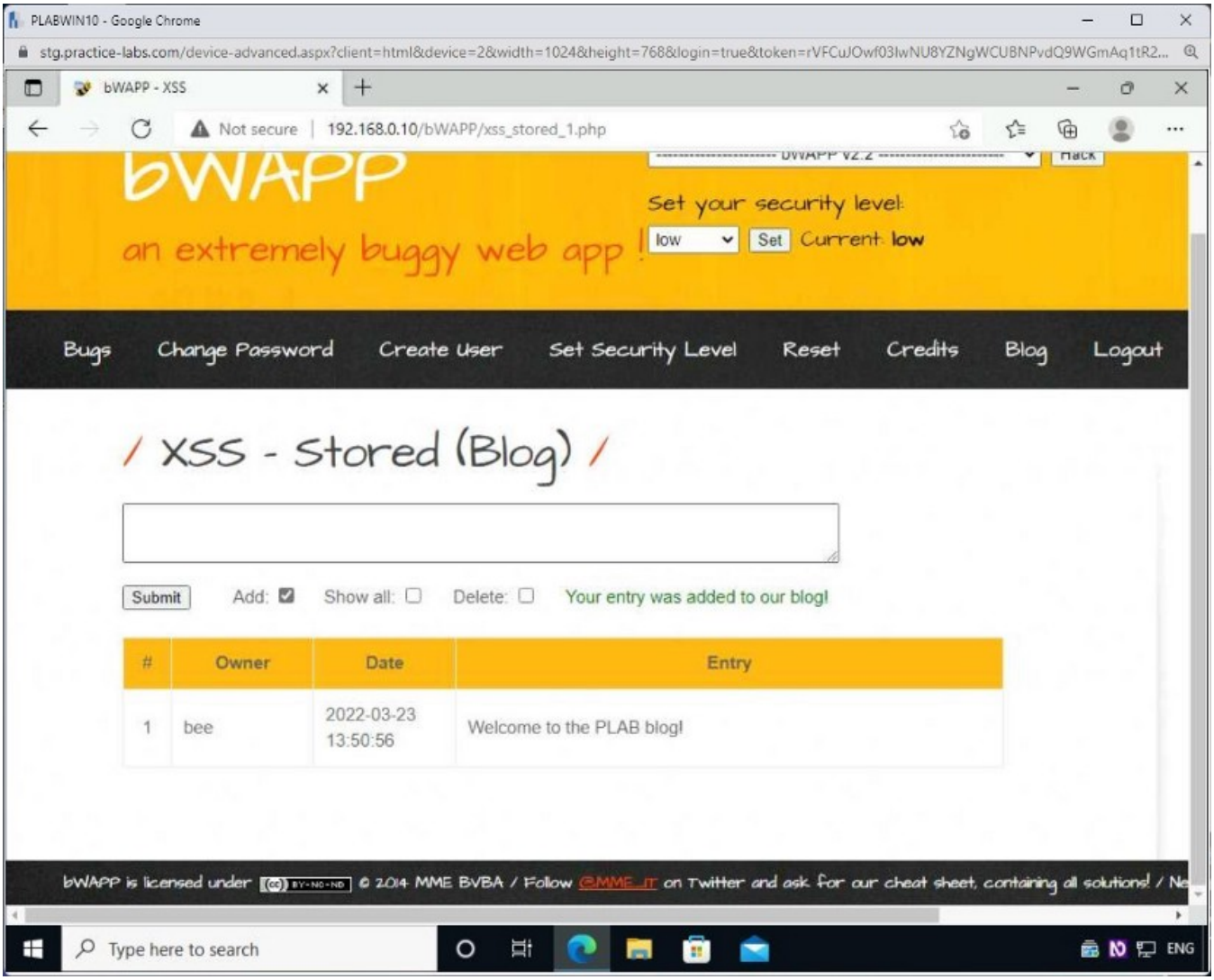
Click **Submit**.



Step 8

Observe the output displayed on the screen.

The entered values are stored in the application's database and are displayed permanently on the application's **XSS — Stored (Blog)** page. This could be a potential entry point for stored XSS attacks.

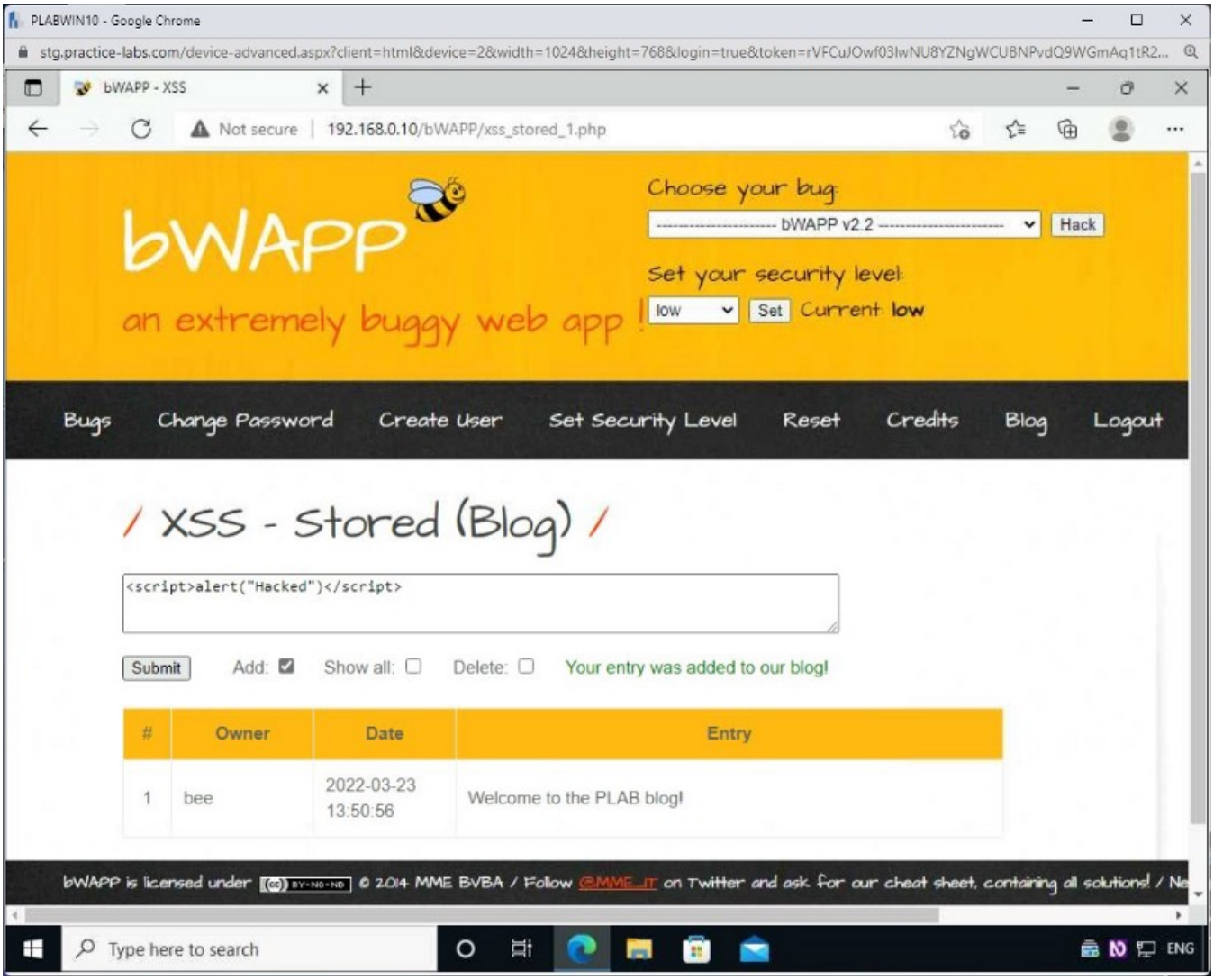


Step 9

On the **XSS — Stored (Blog)** webpage, type the following message in the text box:

```
<script>alert("Hacked")</script>
```

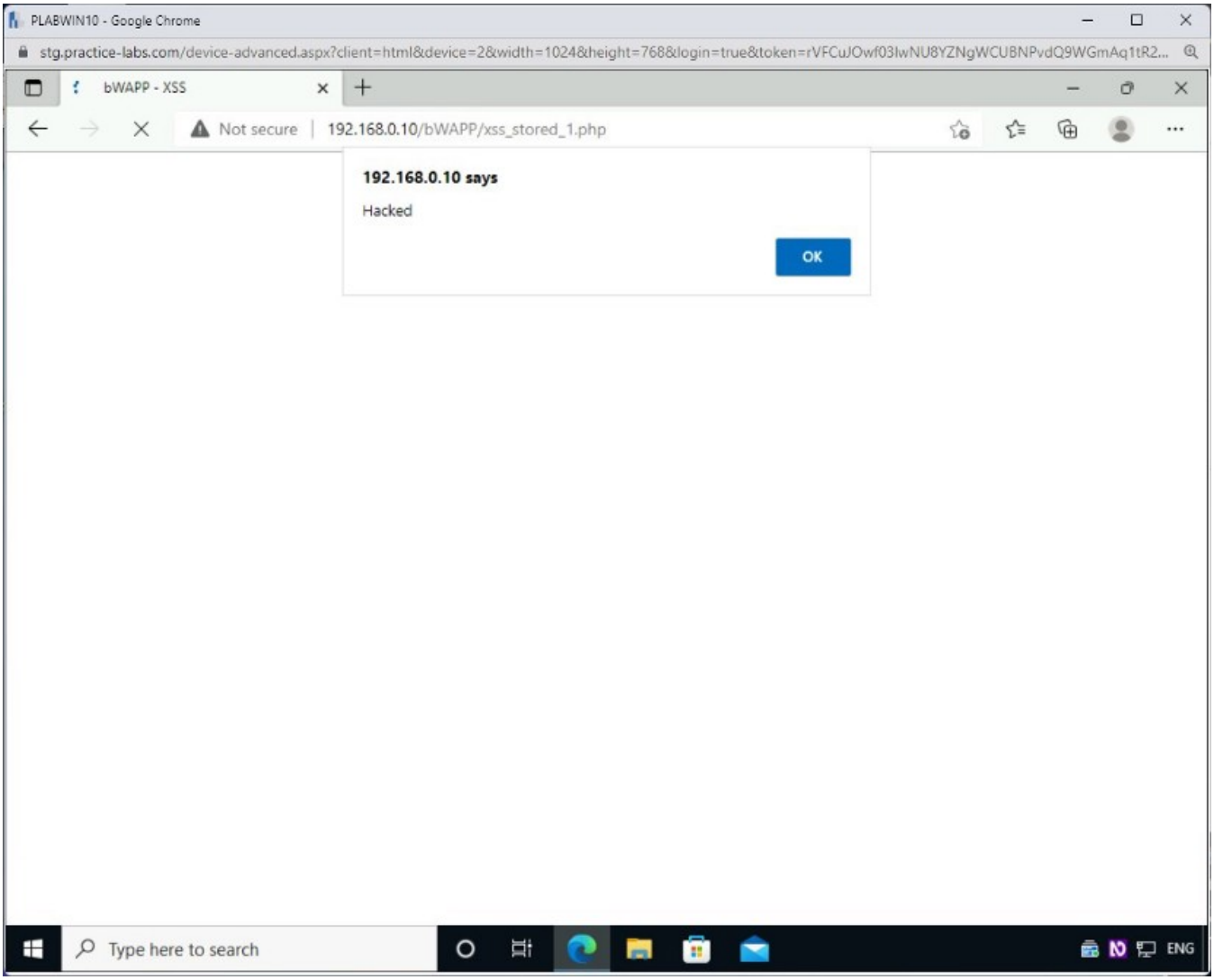
Click **Submit**.



Step 10

The server processes the script and displays the alert message box.

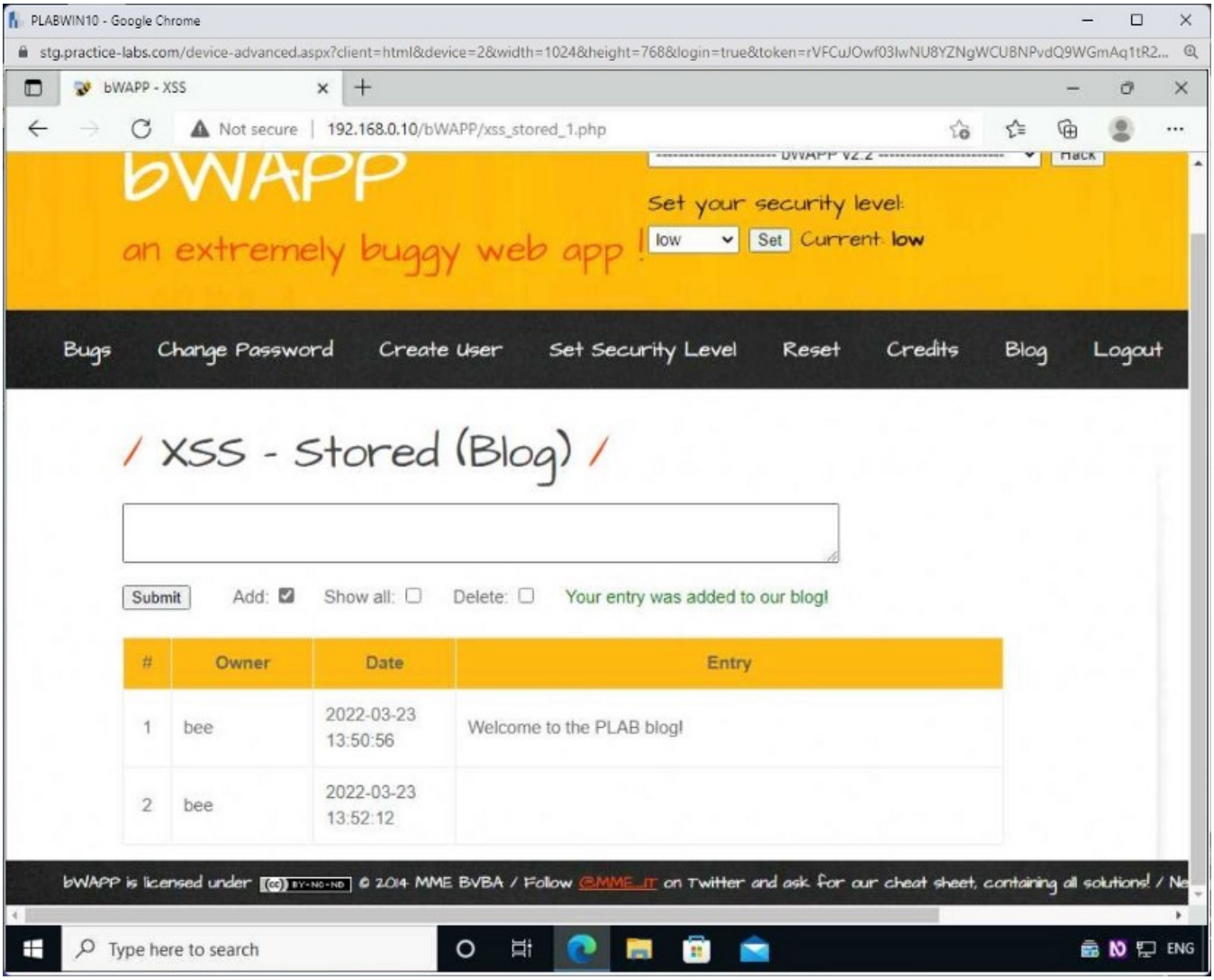
Click **OK**.



Step 11

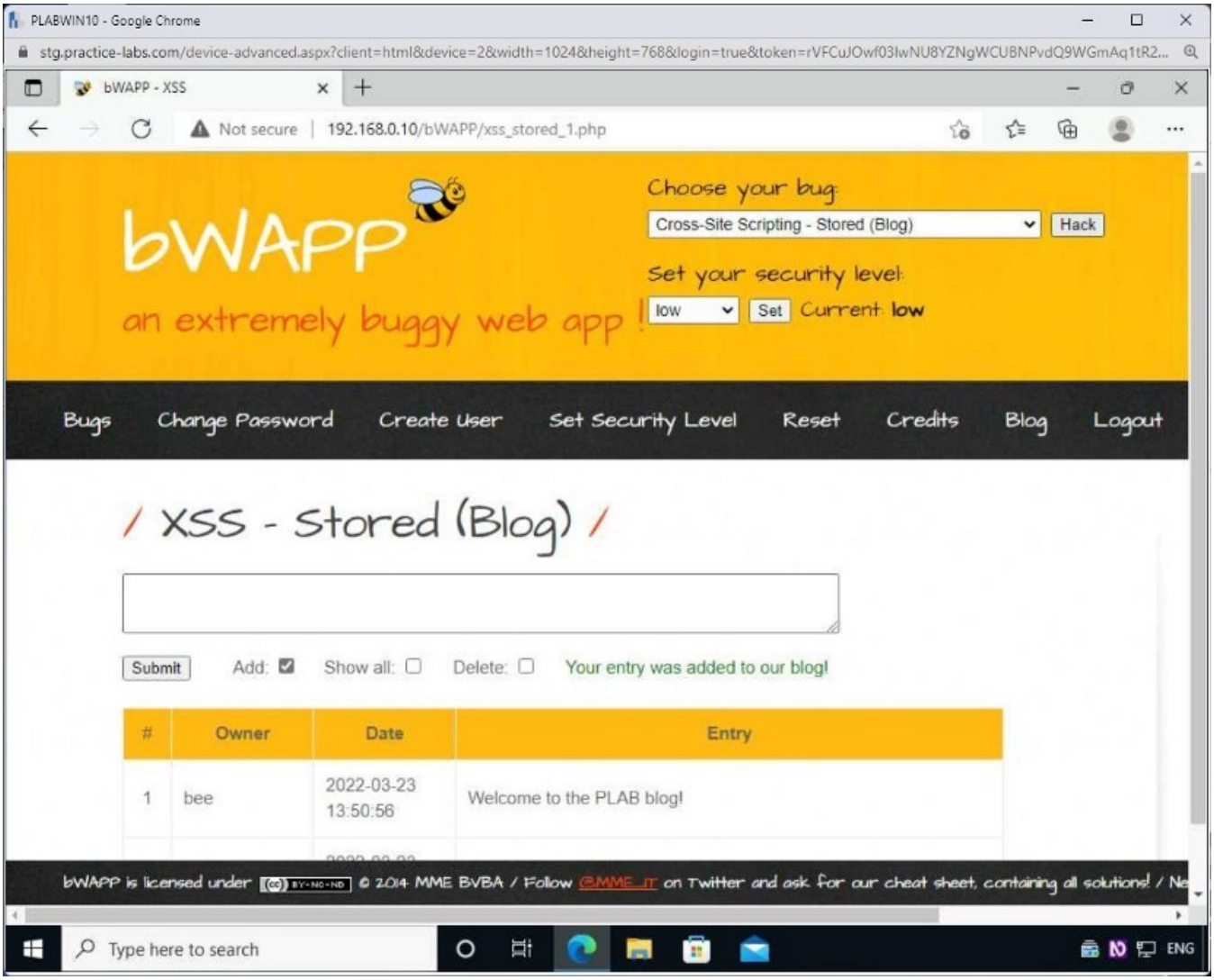
You are back on the **XSS — Stored (Blog)** webpage. Notice that there is a second entry in the list, but it is empty. The entered JavaScript gets reflected in the stored XSS section of the web page. Unlike reflected XSS, stored XSS is permanent since the entered values are stored in the application's database.

Therefore, attackers could use this vulnerability to deface any website by displaying an image using stored XSS, stealing users' cookies, and so on.



Step 12

Once again, select **Cross-Site Scripting — Stored (Blog)** from the **Choose your bug** drop-down and click **Hack**.



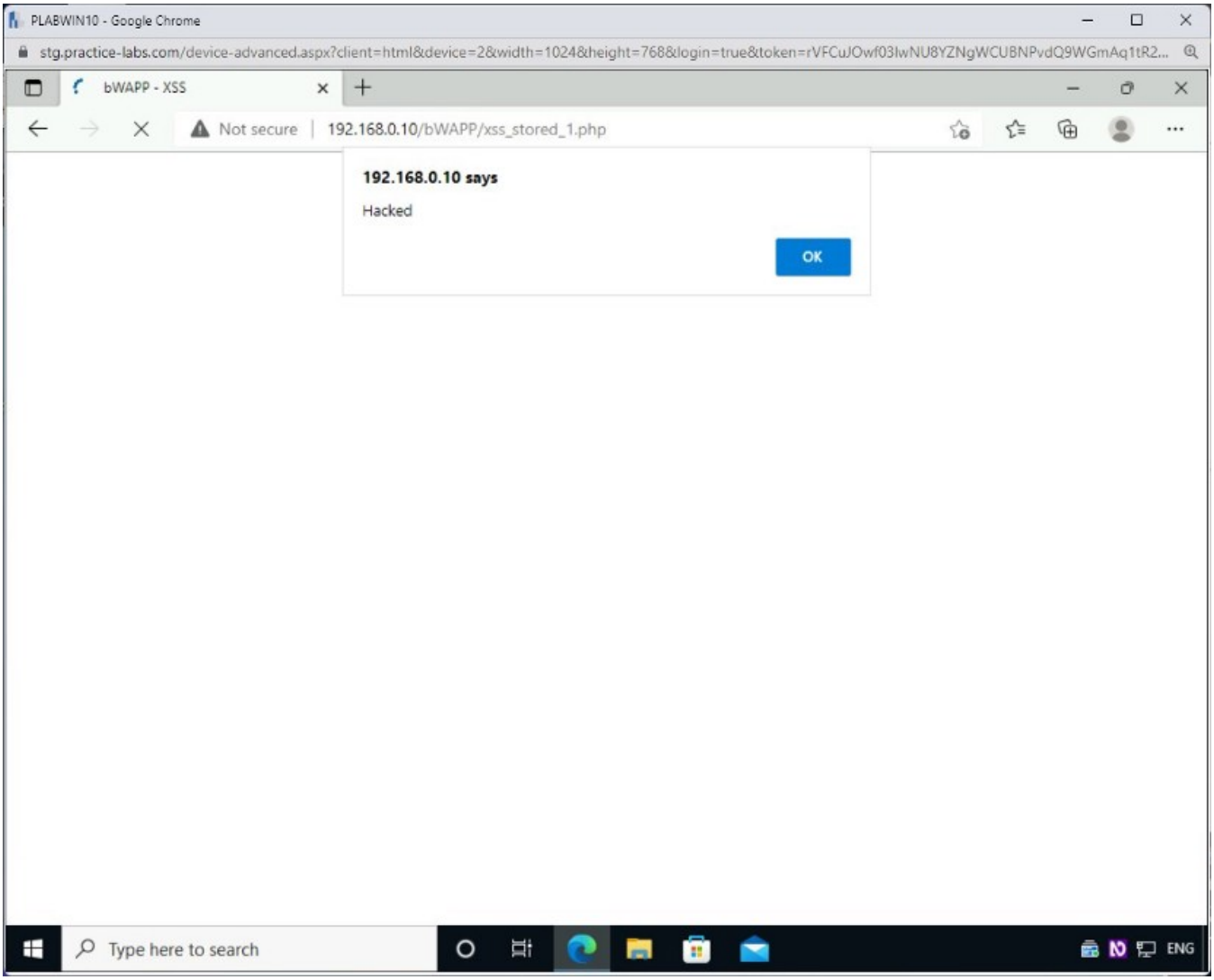
Step 13

The stored XSS script you used on this page is automatically displayed.

Click **OK** in the alert message box.

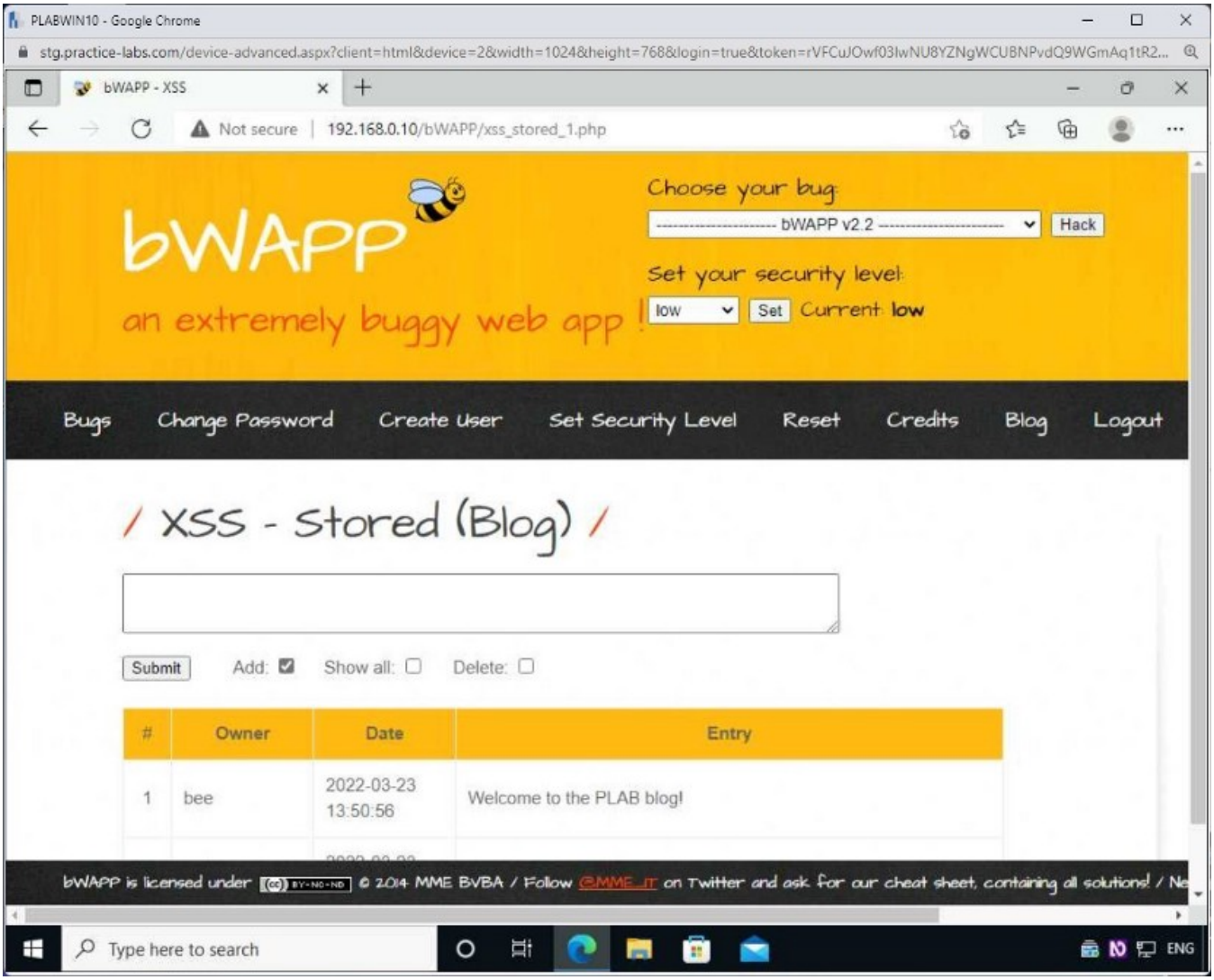
This is because the injected JavaScript is stored in the database, and it gets executed every time you navigate to the stored XSS section.

However, in reflected XSS, this does not happen since the values are not stored in the application's database.



Step 14

The **XSS — Stored (Blog)** webpage is now displayed.



Keep the Microsoft Edge window open.

Task 7 — Perform Cross-site Scripting (XSS) — Reflected (HREF) Attack

In a Reflected (HREF) attack, an attacker can insert a malicious script in the URL itself. This problem occurs when a web page requests a string of information, such as a username, and displays it in the URL. If the input is not sanitized, an attacker can leverage this vulnerability to run a malicious script.

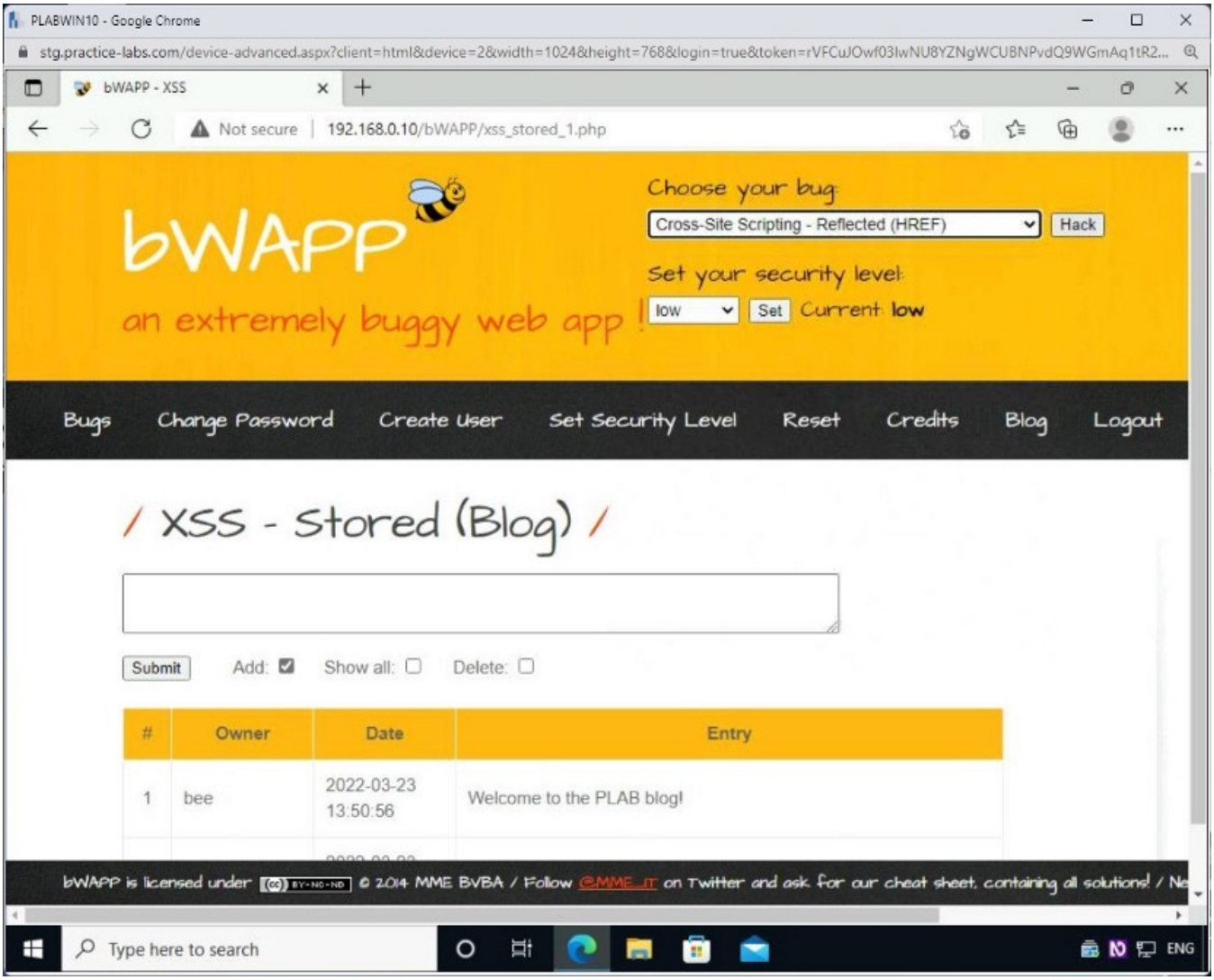
In this task, you will perform an XSS — Reflected (HREF) attack. To do this, perform the following steps:

Step 1

Ensure you have powered on all the devices listed in the introduction and connect to **PLABWIN10**.

The **Microsoft Edge** window should be open.

To begin reflected cross-site scripting attack, from the **Choose your bug** drop-down, click **Cross-site Scripting — Reflected (HREF)** and click **Hack**.

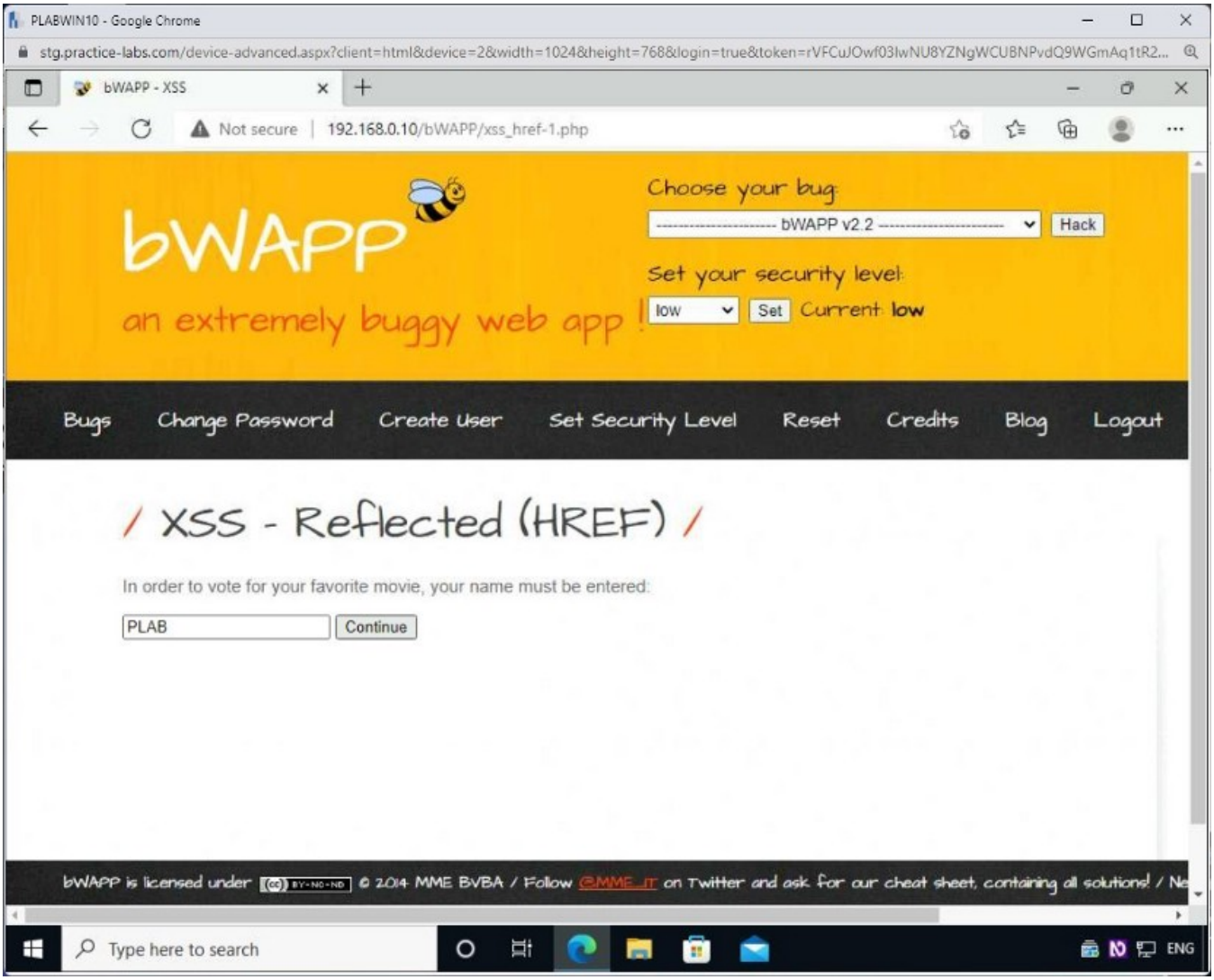


Step 2

The **XSS — Reflected (HREF)** webpage is displayed. In the text box, type the following:

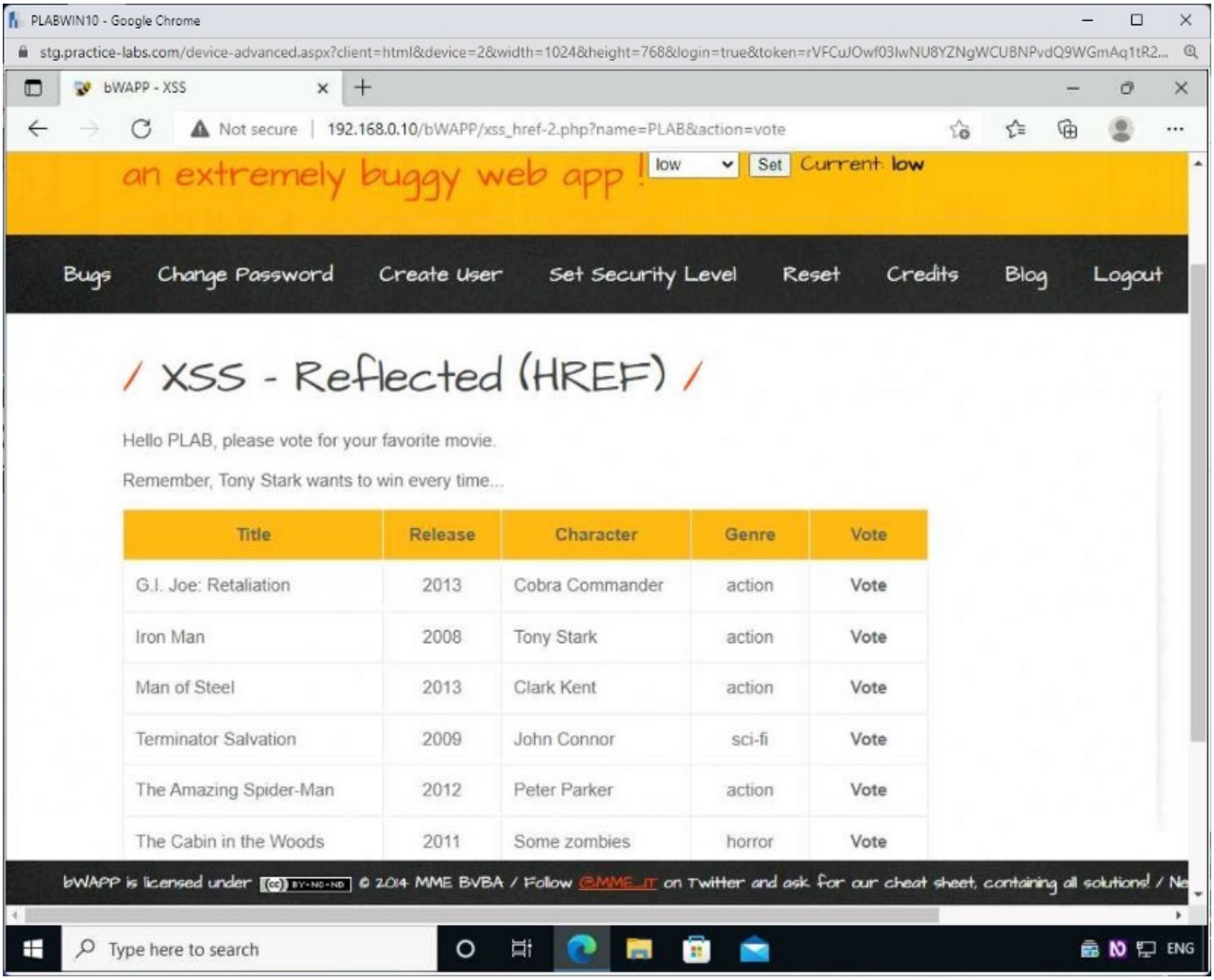
```
PLAB
```

Click **Continue**.



Step 3

Notice that the entered name is now embedded in the URL and displayed on the webpage.

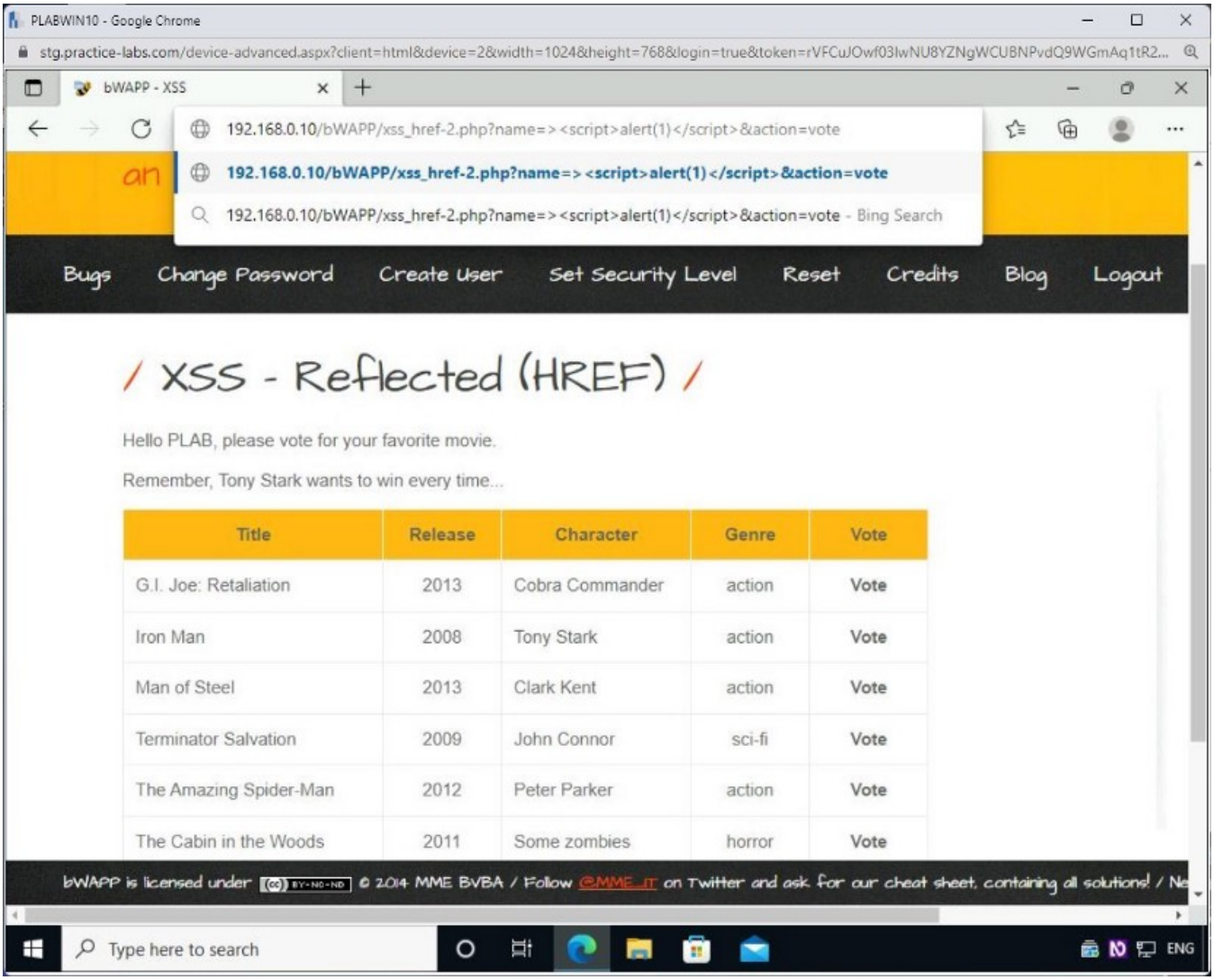


Step 4

In the URL, replace the name **PLAB** with the following script:

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

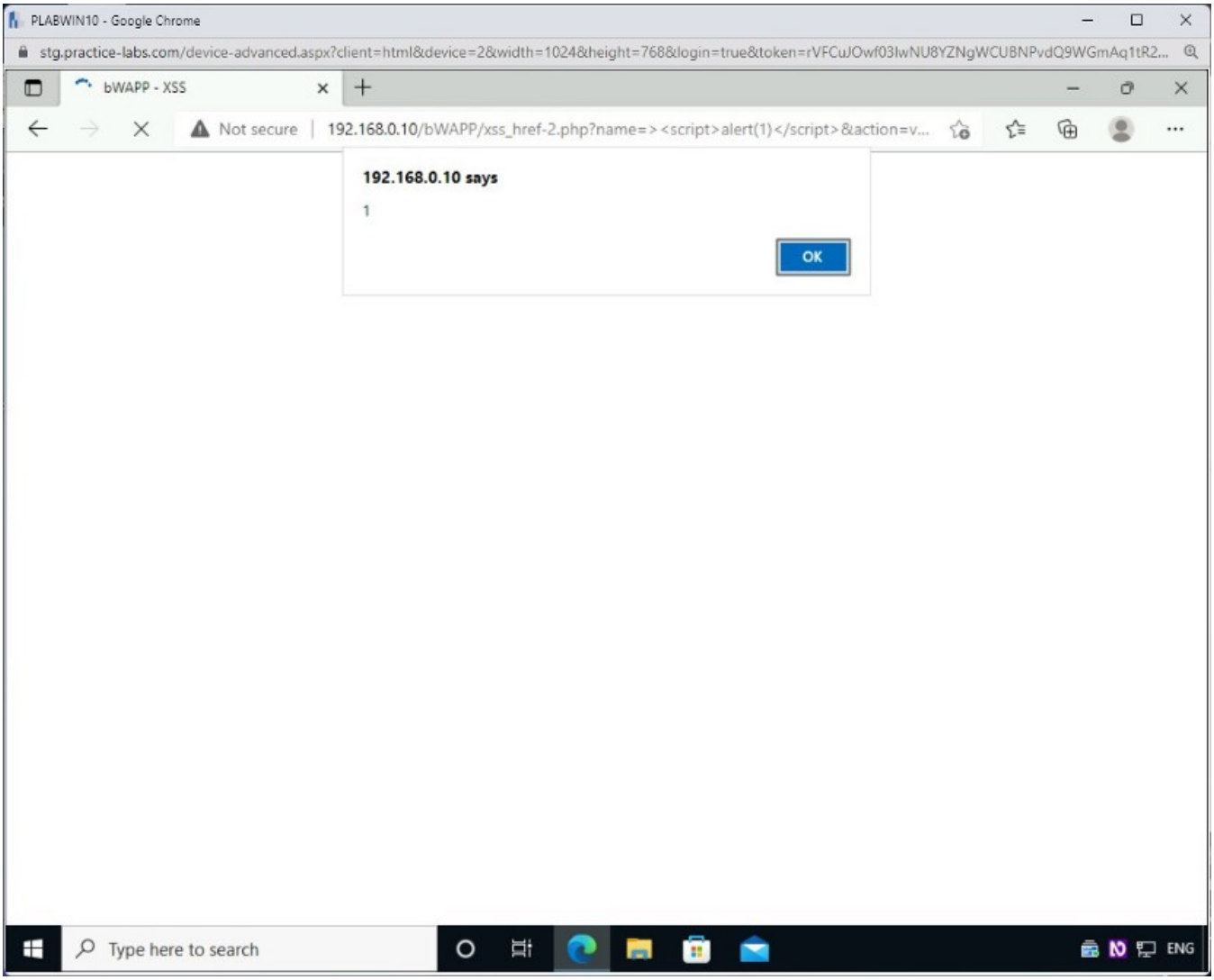
Press **Enter**.



Step 5

You are navigated to another page. An alert is displayed with the value **1**.

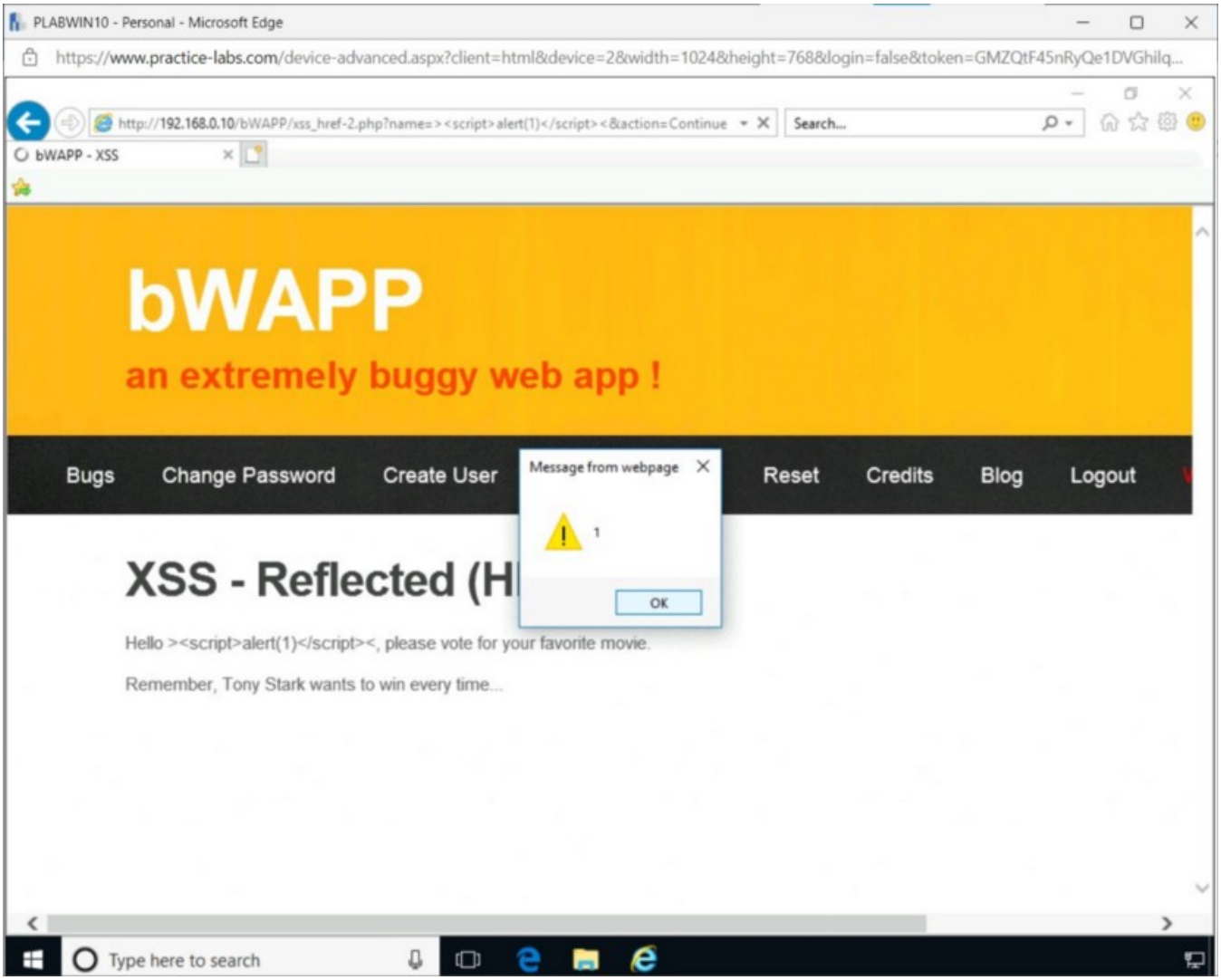
Click **OK**.



Step 6

The alert box continues to display even after you click **OK**.

Alert: If you wanted to return to the page before, you will have to be quick to be able to navigate to another page via the back button.



Exercise 3 — Web Application Hacking Methodology

Every attack follows a specific methodology, nothing but a series of defined steps. Similarly, in web application hacking, hackers follow a methodology covering web applications from all aspects.

In this exercise, you will learn about the web application hacking methodology.

Learning Outcomes

After completing this exercise, you will have further knowledge of:

- Web Application Hacking Methodology Concepts

The Concept of Web Application Hacking Methodology

Attackers use a series of steps to attack web applications. These series of steps are known as web application hacking methodology. The key steps in this methodology are:

- **Footprinting web infrastructure**: is performed using various methods, such as Whois lookup, DNS interrogation, hidden content discovery, detecting firewalls and load balancers, and port scanning.
- **Analyzing target web applications**: is performed using various methods, such as identifying the server-side and endpoint technologies. It also includes identifying server-side functionalities, files and directories, web application vulnerabilities, and identifying the attack surface.
- **Bypassing client-side controls:** is performed by attacking hidden form fields and web browser extensions, identifying the source code flaws by reviewing them and evading XSS filters.
- **Attacking authentication methods**: is performed by bypassing authentication, performing username enumeration, cookie exploitation, password and session attacks.
- **Attacking authorization schemes**: is compromising an attack and then escalating privileges. It is also performed by manipulating HTTP requests.
- **Attacking access controls**: includes an attack on applications using different user accounts, attacking a low-privileged user account, then escalating privileges, and attacking the static resources mentioned in the URL
- **Attacking session management controls**: are attacks such as man-in-the-middle, session replay, and session hijacking attacks.
- **Conducting injection attacks**: are attacks such as OS command, SMTP, SQL, LDAP, XPath, and file injection.
- **Conducting application logic flaw attacks**: includes logic flows, programming errors, and misconfiguration exploitation.
- **Conducting shared environment attacks**: includes attacks on access methods and launching attacks by running scripts from one application to another one.
- **Conducting database connectivity attacks**: includes various attacks, such as connection pool DoS, connection string injection, and Connection String Parameter Pollution (CSPP) attacks.
- **Conducting web application client attacks**: includes various attacks, such as cross-site scripting, HTTP header injection, request forgery, redirection, frame

injection, and session fixation.

- **Conduct web service attacks**: includes various attacks, such as SOAP and XML injection, WSDL probing, application logic, and database attacks.

Exercise 4 — Footprint and Analyze Web Infrastructure

The web application attack methodology defines an attacker who starts the attack by footprinting and analyzing the web infrastructure and applications. It is required that an attacker has information regarding the web infrastructure to devise a correct plan for proceeding further.

In this exercise, you will learn to footprint and analyze the web infrastructure.

Learning Outcomes

After completing this exercise, you will be able to:

- Web Applications Enumeration Using Wafw00f
- Perform website Enumeration using Nmap
- Detect the IP addresses Using the Host Command
- Information Gathering Using Legion

Task 1 — Enumerate web Applications Using Wafw00f

A web application enumeration can be prevented using a web Application Firewall (WAF). The wafw00f tool helps determine whether the web application is behind a WAF.

A hacker may want to analyze the web application before launching an attack. wafw00f is a useful tool to determine whether the application is behind a firewall. Accordingly, an attacker may decide the next course of action.

In this task, you will learn to use wafw00f. To do this, perform the following steps:

Step 1

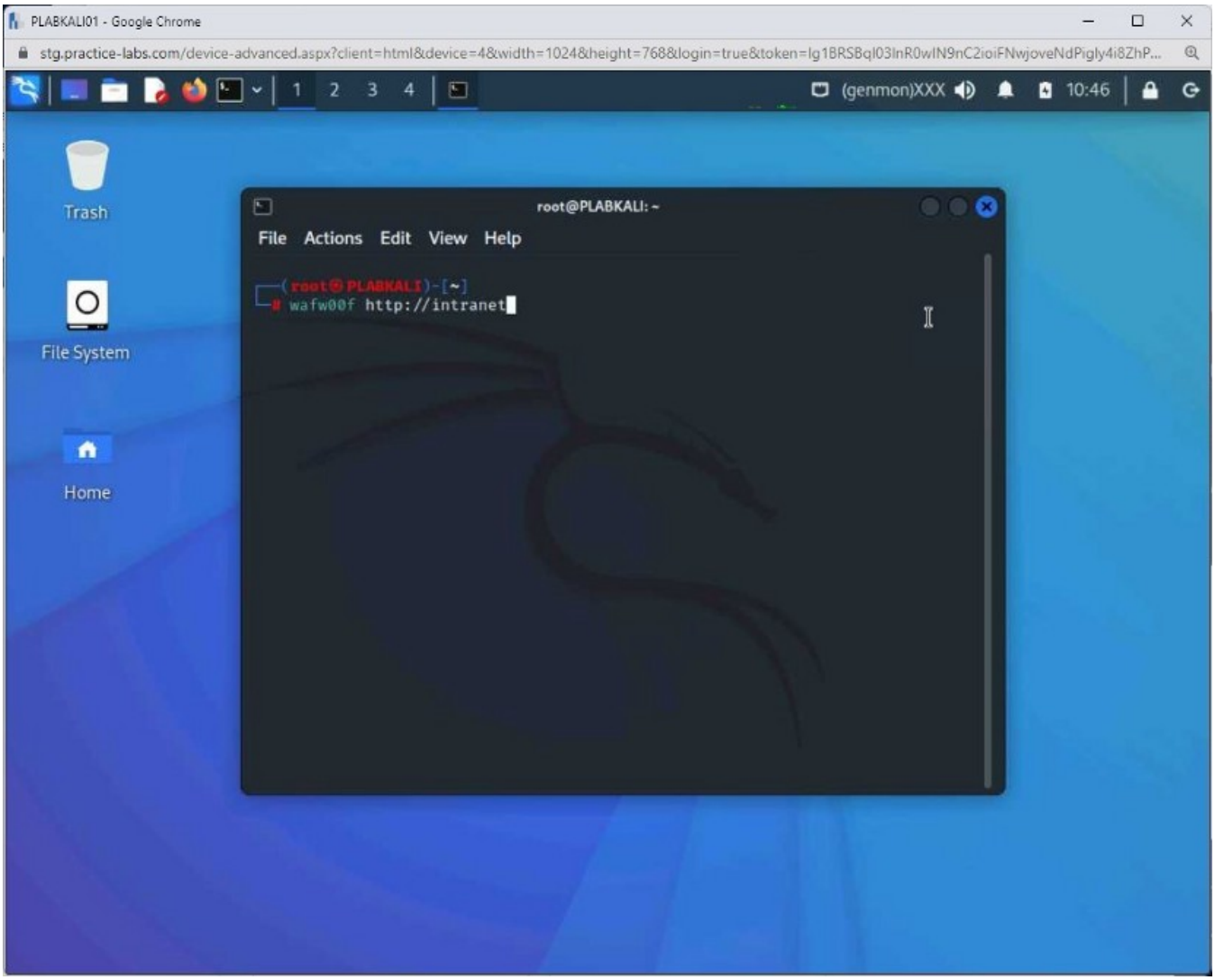
Ensure you have powered on all the devices listed in the introduction and connect to **PLABKALI01**.

You will attempt to determine whether a web application is behind the **web Application Firewall (WAF**).

You will use a tool named **wafwoof** for this purpose. Type the following command:

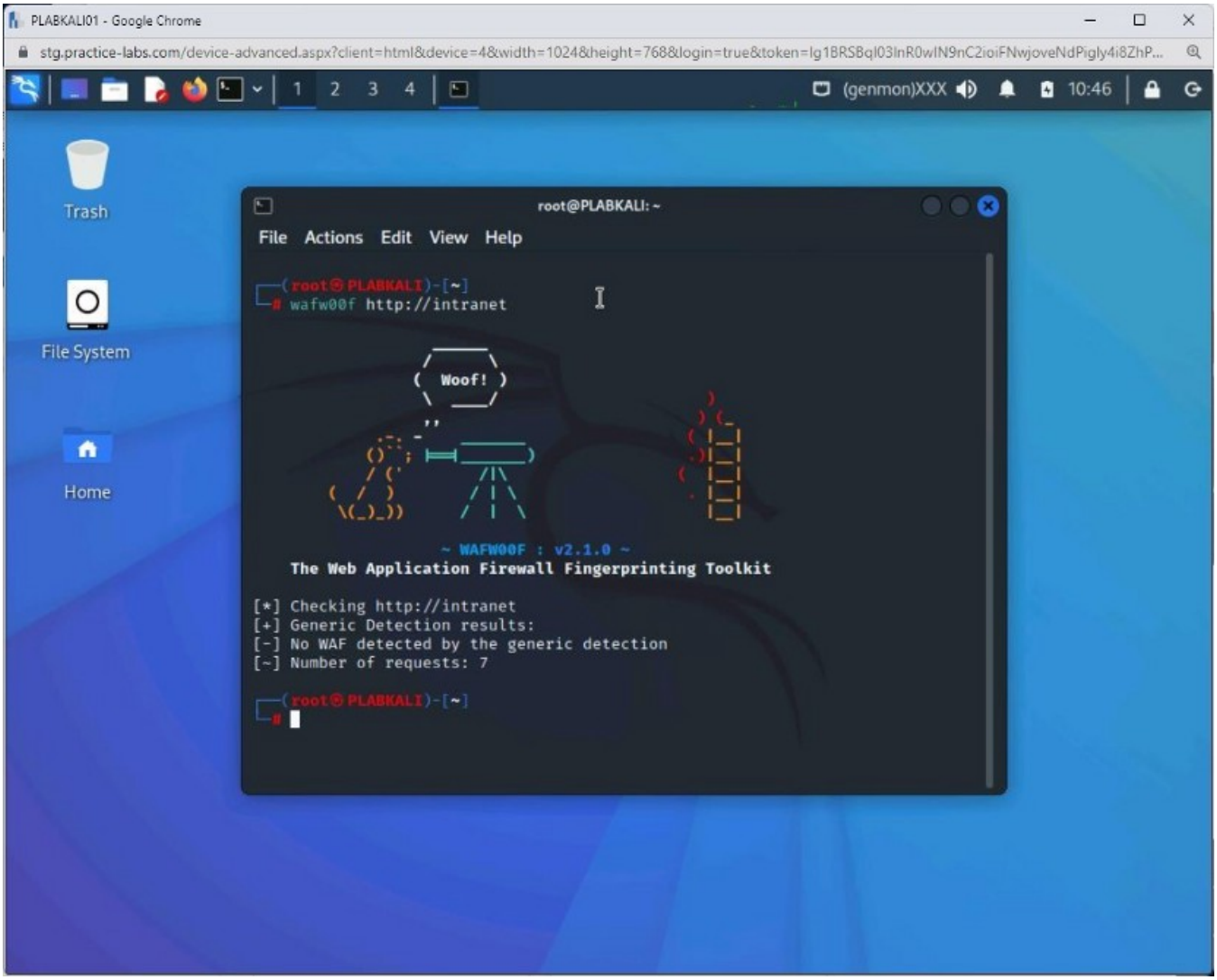
```
wafw00f http://intranet
```

Press **Enter**.



Step 2

Notice that **wafwoof** did not detect a **WAF**.



Keep the terminal window open.

Task 2 — Perform Website Enumeration using Nmap

There are different methods to enumerate a website. For example, you can use a manual method using a web browser. You can try:

```
http://www.plab.com/admin
```

After the URL, you can add a directory name, such as admin. You are likely to get one of the following responses:

- **200** — OK
- **401** — Unauthorized
- **402** — Payment Required

- **403** — Forbidden
- **404** — Not Found

If the admin does not return a **404** error but something else, such as **403**, it indicates clearly that this directory exists.

You can also enumerate a website using Nmap, which provides several scripts to enumerate different types of websites, such as WordPress or Drupal.

In this task, you will perform website enumeration using Nmap. To do this, perform the following steps:

Step 1

Connect to **PLABKALI01**.

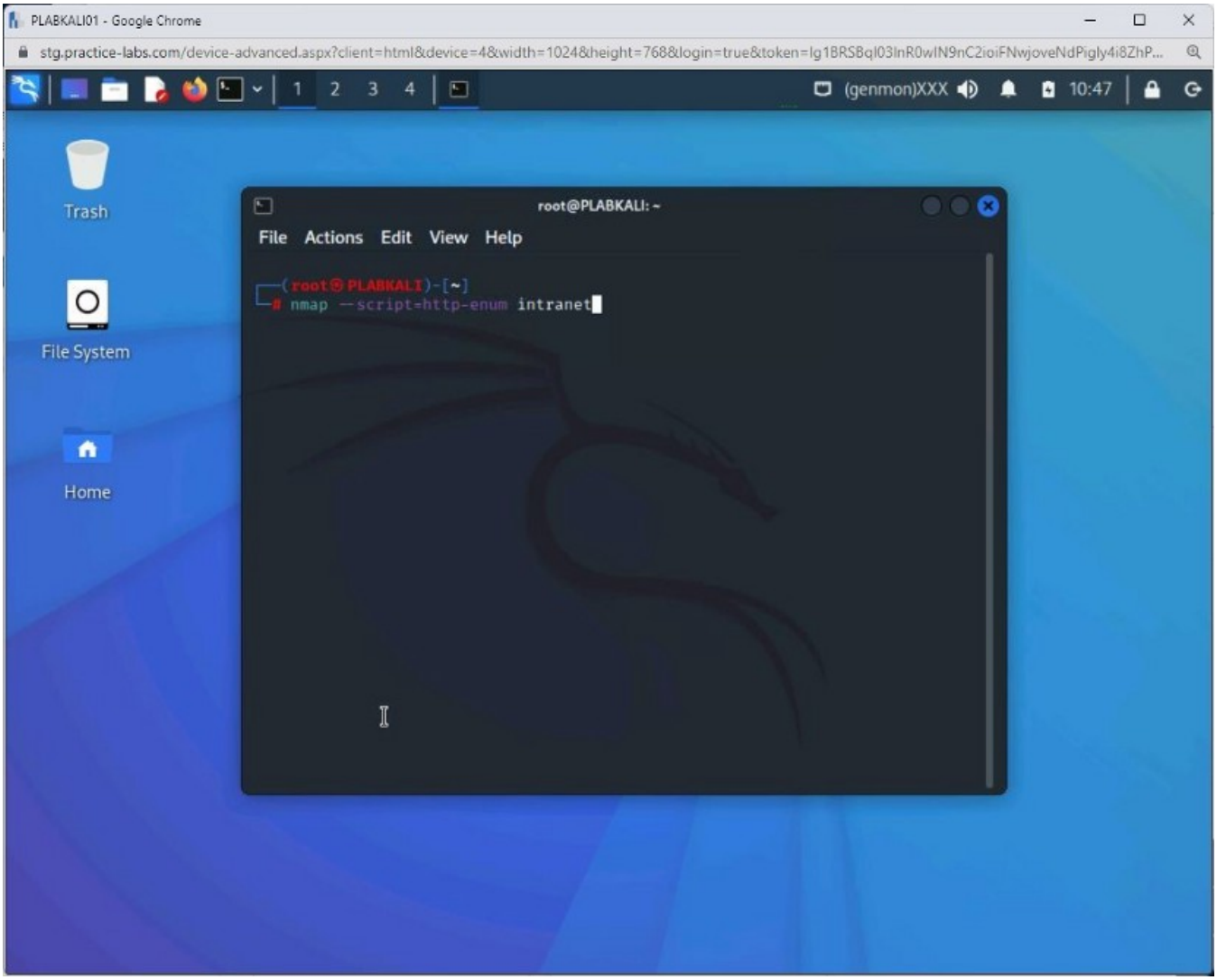
Clear the screen by entering the following command:

```
clear
```

To perform a website enumeration, type the following command:

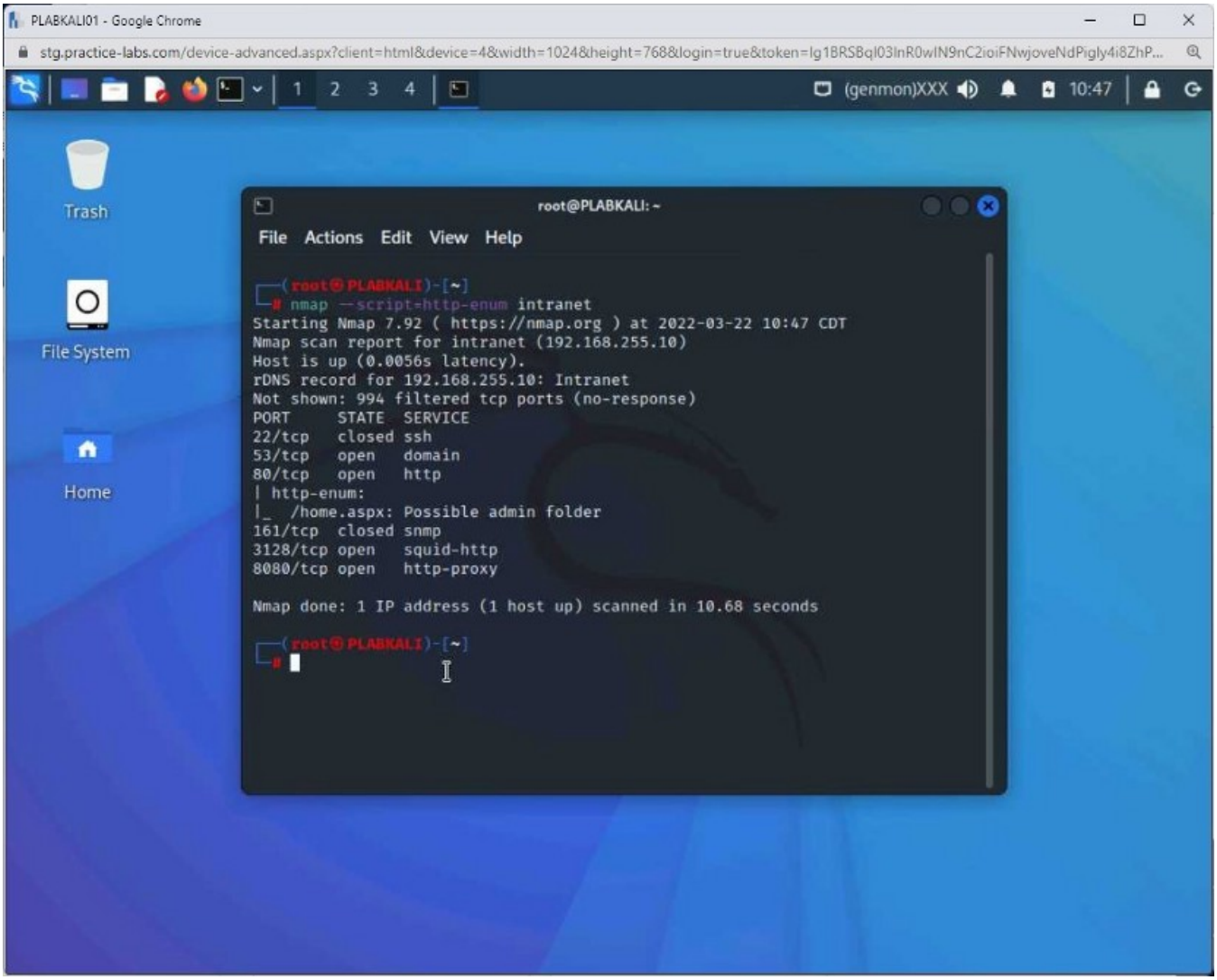
```
nmap --script=http-enum intranet
```

Press **Enter**.



Step 2

Notice the output. It has been able to list the open ports and a possible admin folder.



Keep the terminal window open.

Task 3 — Detect the IP addresses Using the Host Command

Using the host command, an attacker can detect the IP addresses of the web application. If more than one IP address is assigned to the web application, it can be detected. It is also useful to determine if the web application uses a load balancer, taking many web applications. If an attacker is planning to conduct a DoS attack, this command can determine a load balancer.

To use the host command, perform the following steps:

Step 1

Connect to **PLABKALI01**.

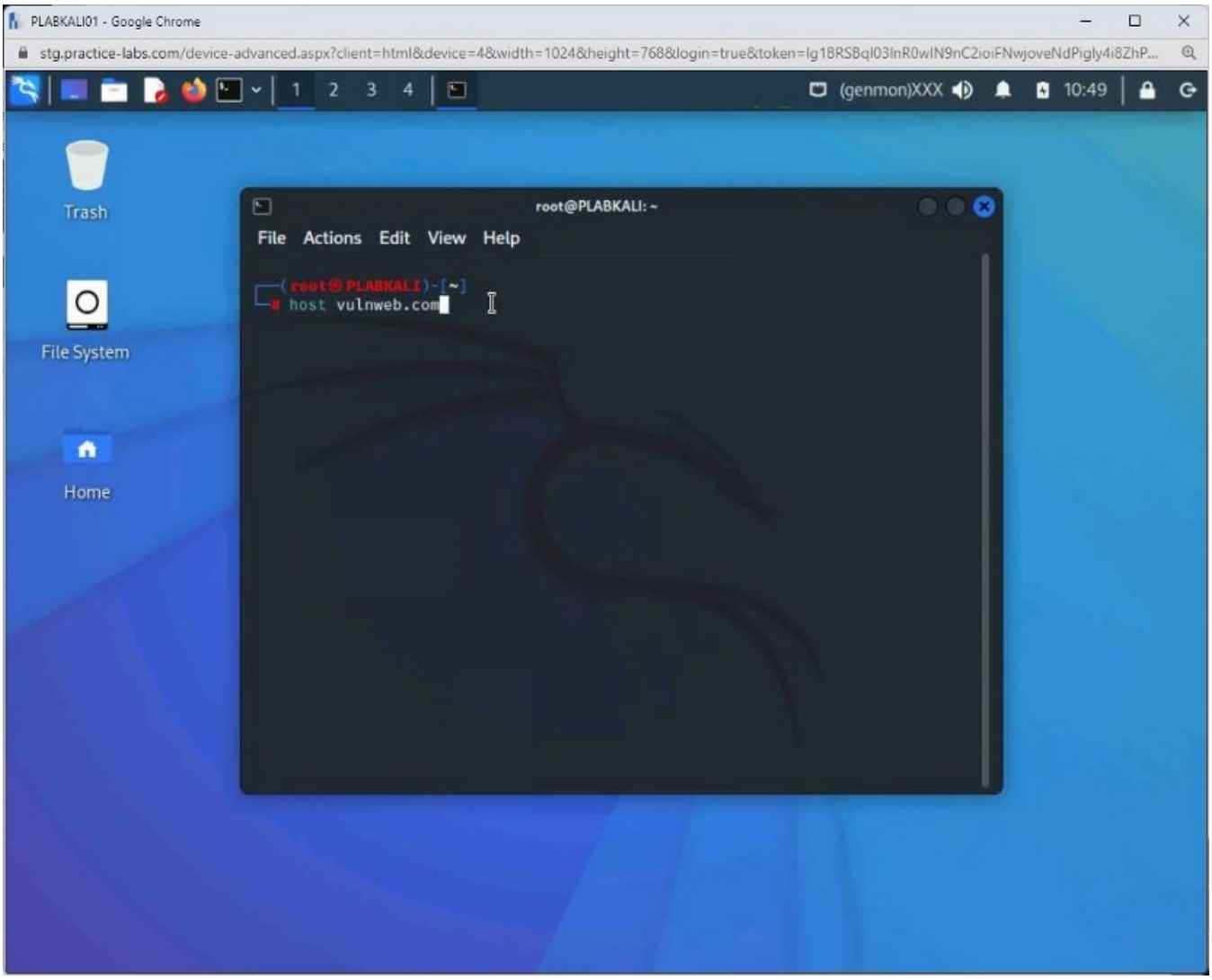
Clear the screen by entering the following command:

```
clear
```

To determine the IP address(es), type the following command:

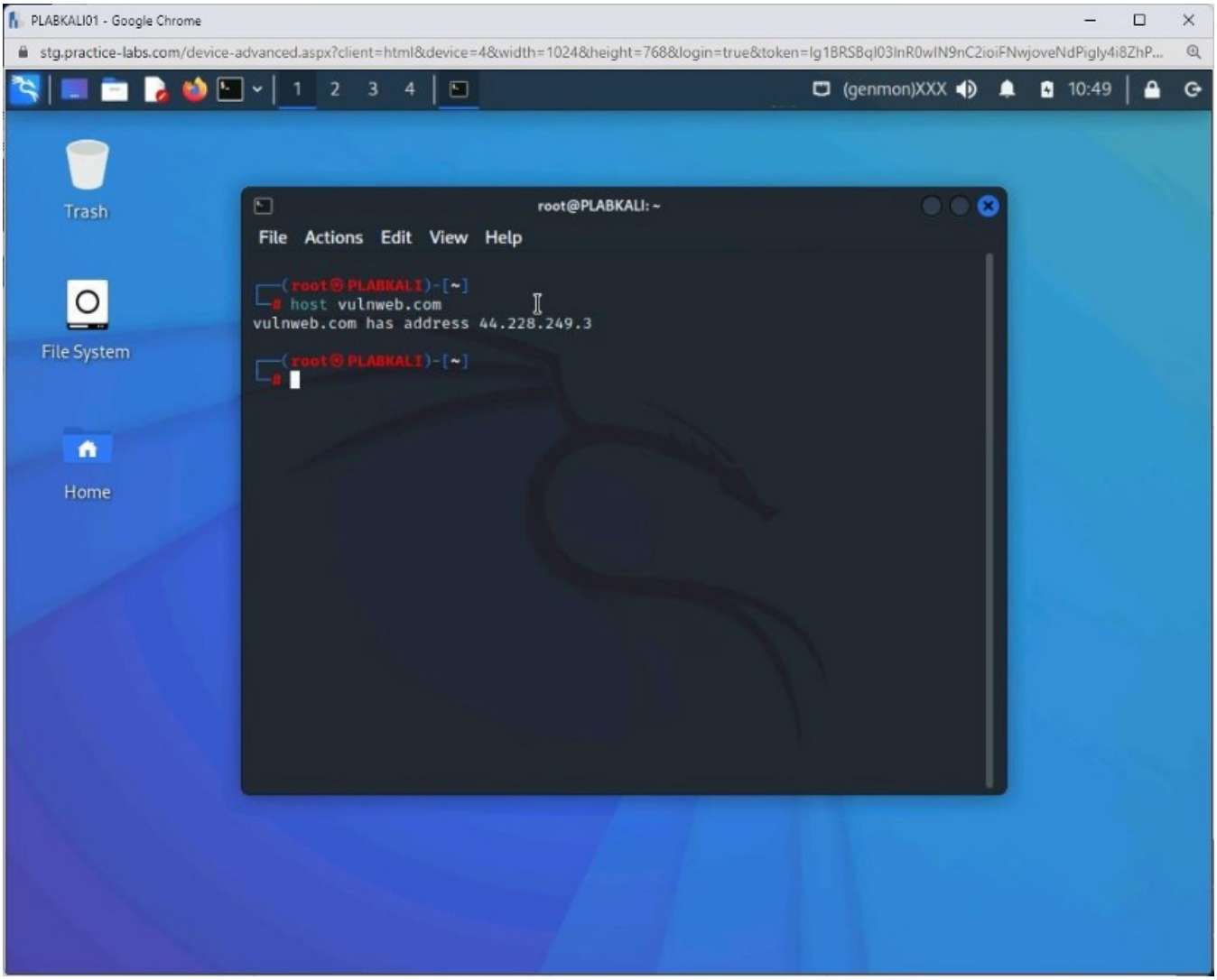
```
host vulnweb.com
```

Press **Enter**.



Step 2

The output shows only one IP address, which means the web application is not using a load balancer.



Keep the terminal window open.

Task 4 — Information Gathering Using Legion

Legion is a Python-based tool with several capabilities, such as vulnerability scanning and information gathering. It uses several tools, such as Nmap, Telnet, and Nikto.

To perform information gathering using Legion, perform the following steps:

Step 1

Connect to **PLABKALI01**.

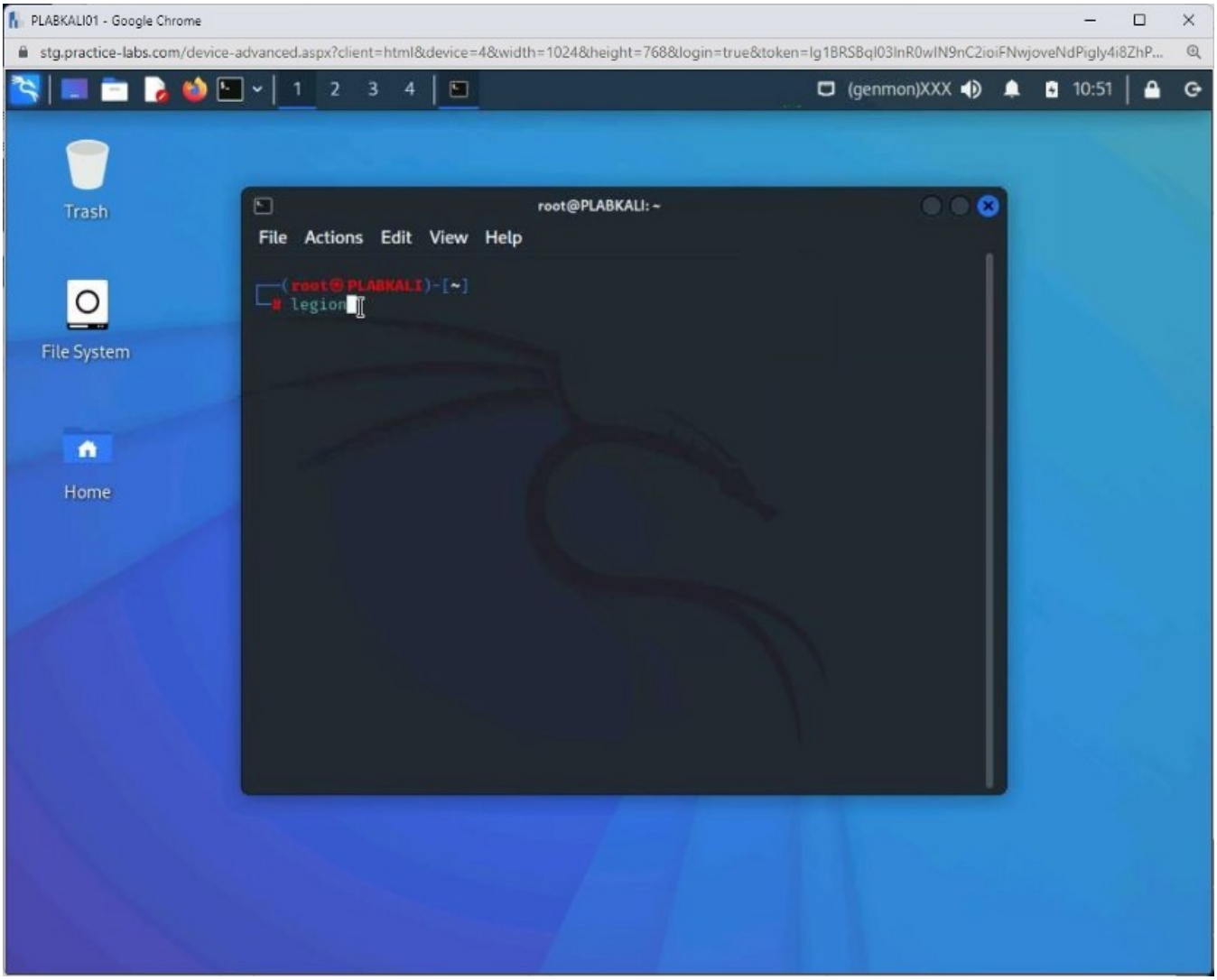
Clear the screen by entering the following command:

```
clear
```

To start Legion, type the following command:

```
legion
```

Press **Enter**.

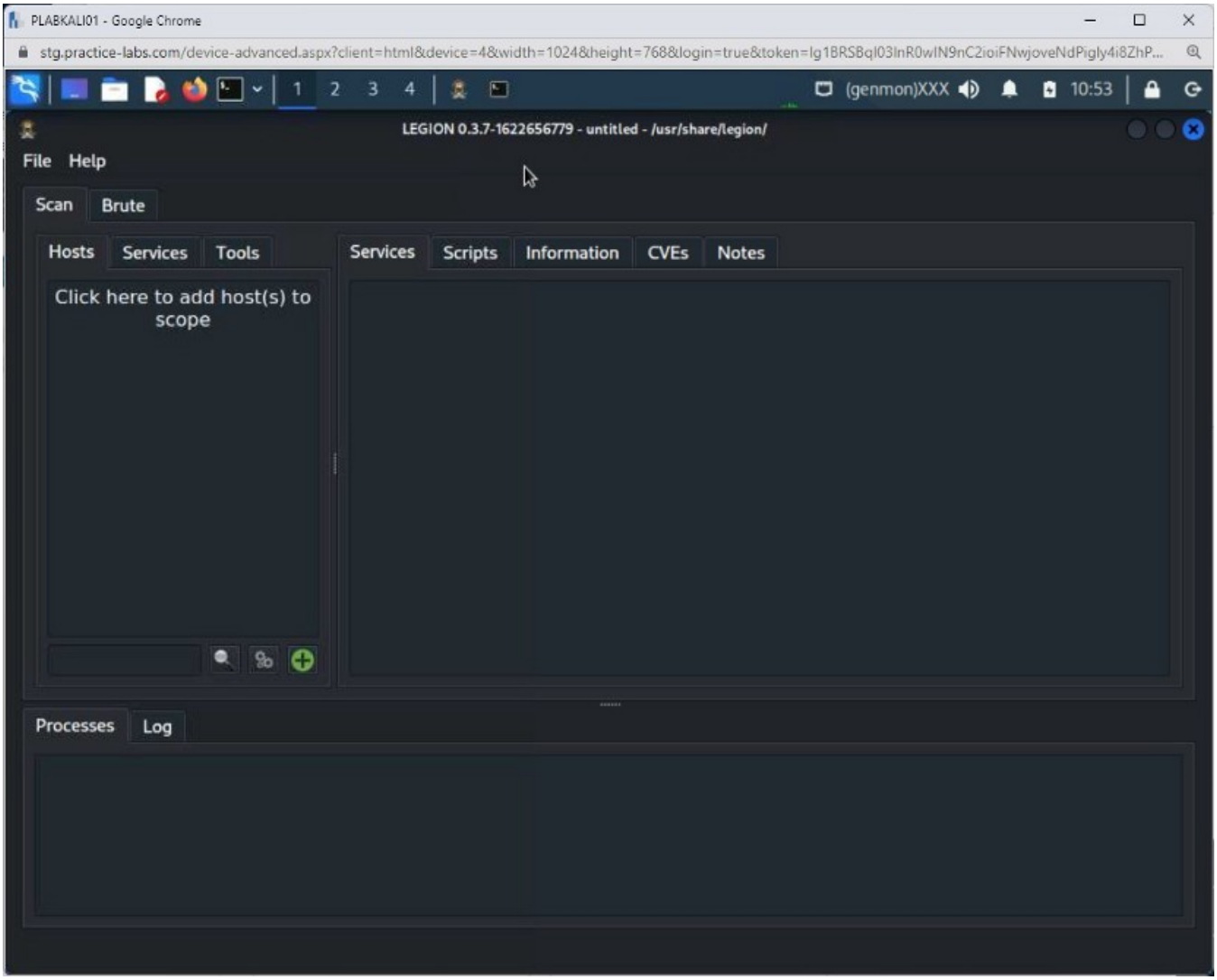


Step 2

The graphical interface of **Legion** is displayed. On the **Hosts** tab in the left pane, click in the **Click here to add host(s) to scope** textbox.

Notice that the right pane currently has four tabs: **Services**, **Scripts**, **Information**, and **Notes**.

***Note**: You may have to manually resize the screen inside of the lab environment.*

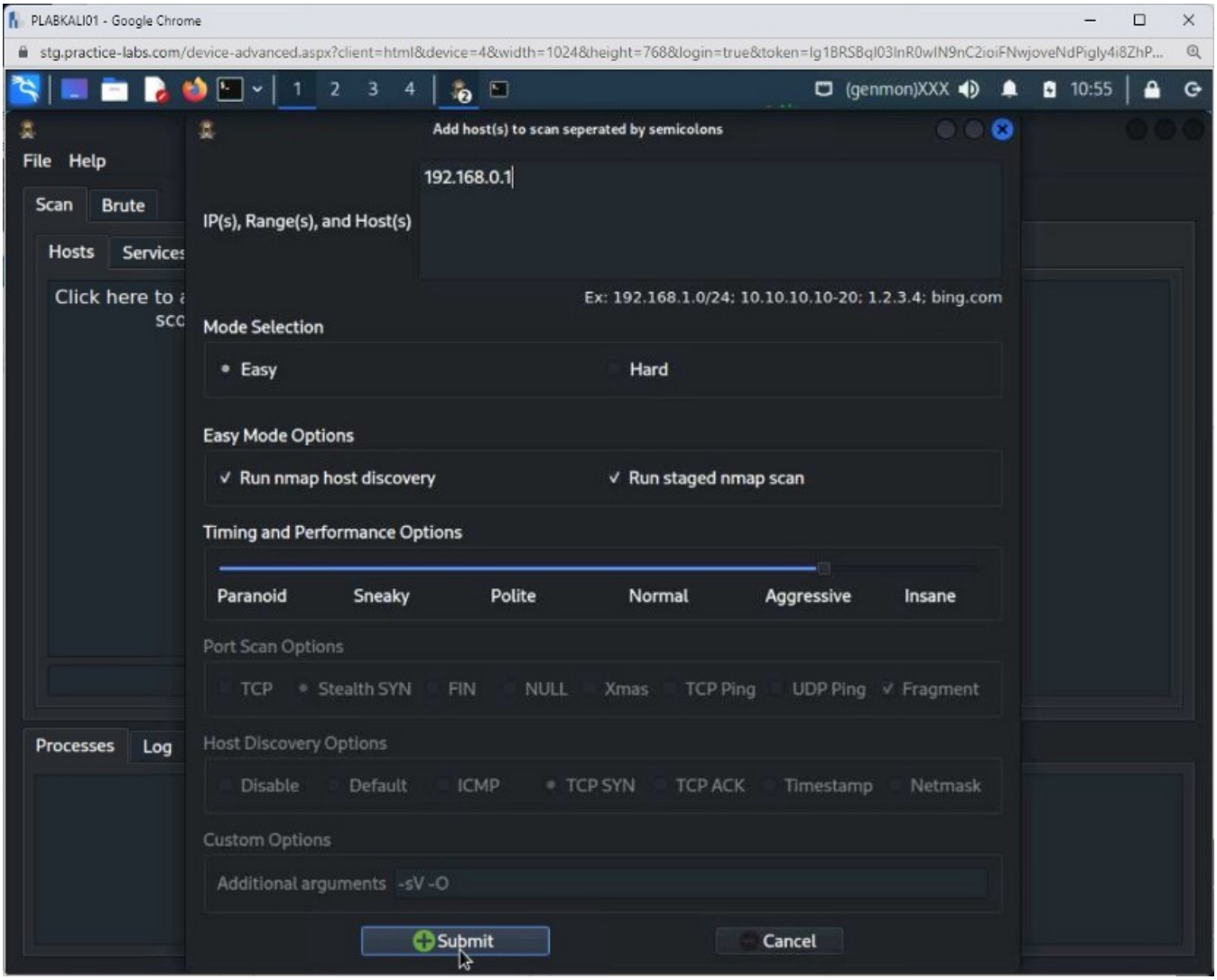


Step 3

The **Add host(s) to scope** dialog box is displayed. In the **IP Range** text box, type the following IP address:

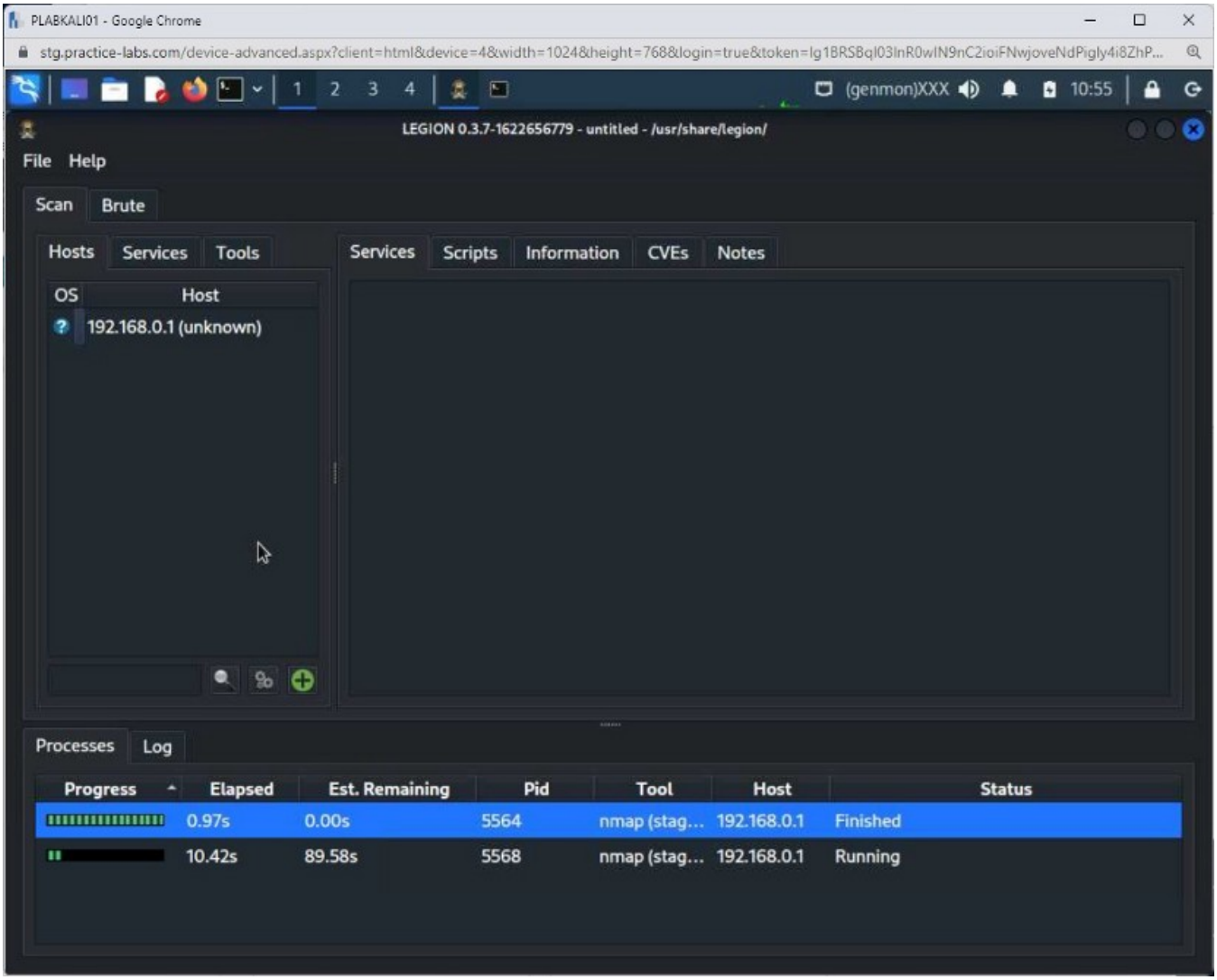
```
192.168.0.1
```

Keep the default selection of the remaining options and click **Submit**.



Step 4

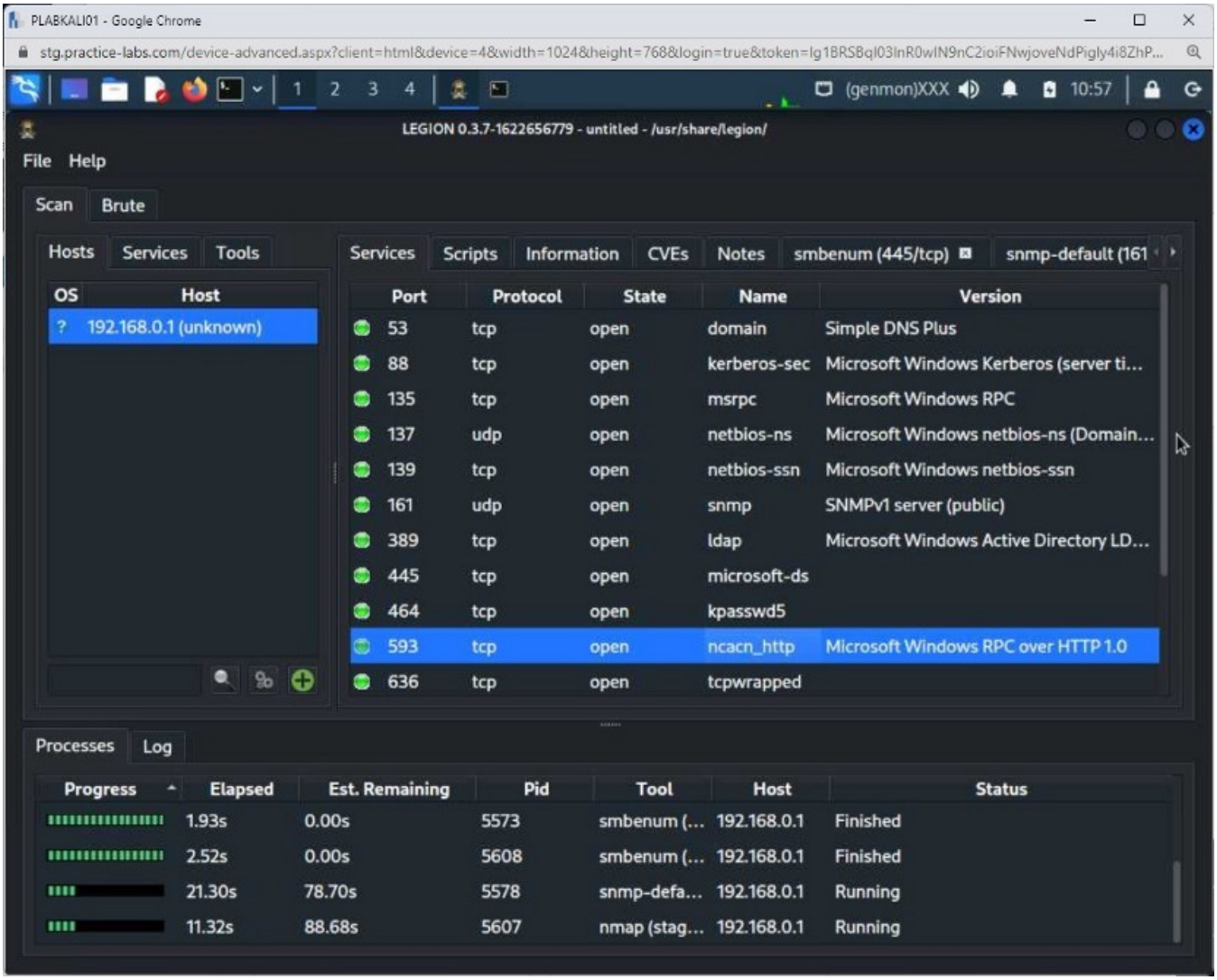
The bottom pane shows the scan progress, which takes a while to finish.



Step 5

Notice that the new tabs are now being added to the right pane. Open ports and running services list are now populated in the Services tab.

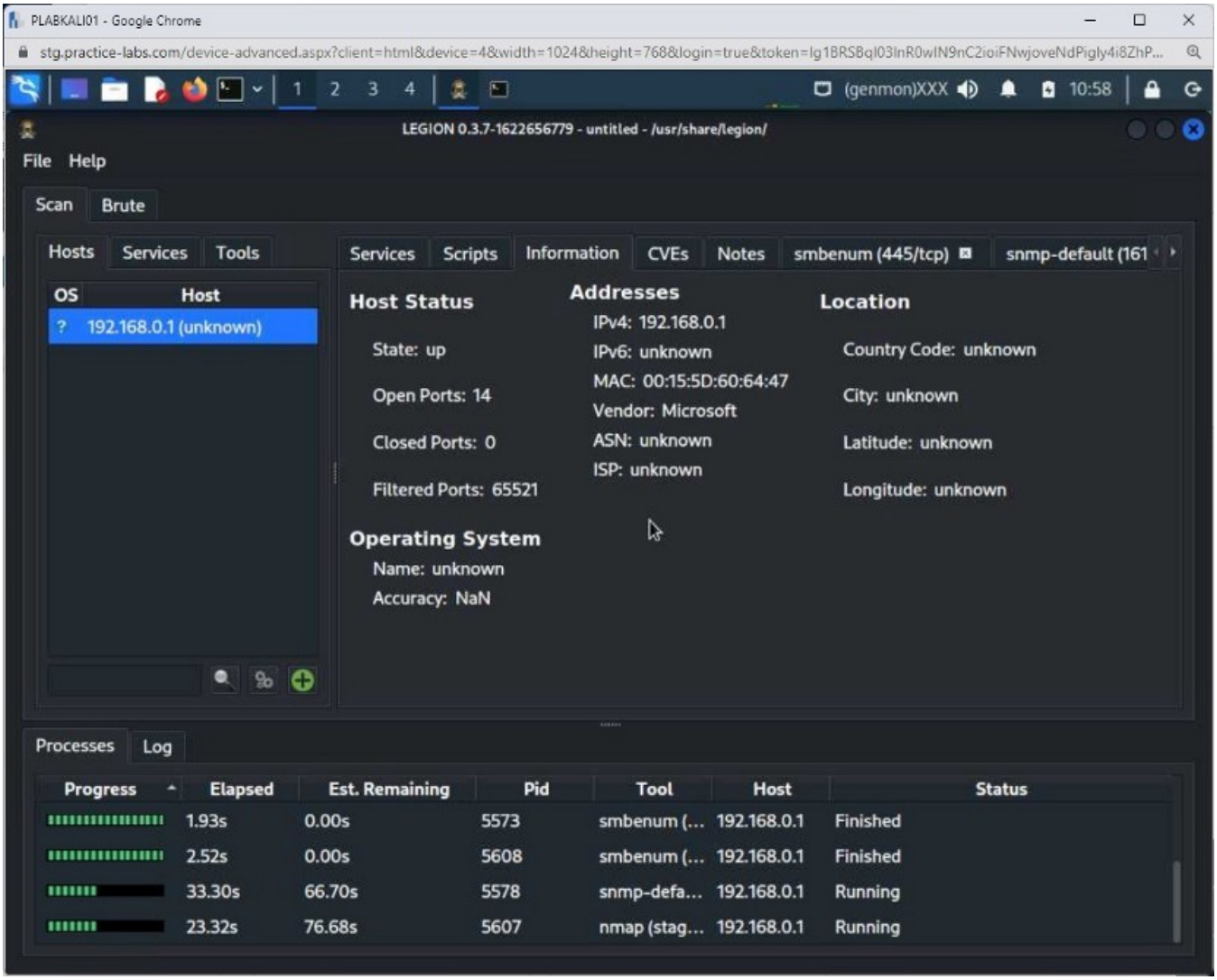
It now only shows the open ports, but also the version numbers. This information can be a goldmine for an attacker.



Step 6

Click on the **Information** tab in the right pane. Notice that it displays information in three sections: **Host Status**, **Addresses**, and **Operating System**.

***Note**: You can also click on each tab after the notes section to see the results of each scan.*


PLABKALI01 - Google Chrome
stg.practice-labs.com/device-advanced.aspx?client=html&device=4&width=1024&height=768&login=true&token=Ig1BRSBqI03InR0wIN9nC2ioiFNwjoveNdPigly4i8ZhP...
1 2 3 4
(genmon)XXX
10:58
LEGION 0.3.7-1622656779 - untitled - /usr/share/legion/
File Help
Scan Brute
Hosts Services Tools
OS Host
? 192.168.0.1 (unknown)
Services Scripts Information CVEs Notes smbenum (445/tcp) snmp-default (161
Host Status
State: up
Open Ports: 14
Closed Ports: 0
Filtered Ports: 65521
Operating System
Name: unknown
Accuracy: NaN
Addresses
IPv4: 192.168.0.1
IPv6: unknown
MAC: 00:15:5D:60:64:47
Vendor: Microsoft
ASN: unknown
ISP: unknown
Location
Country Code: unknown
City: unknown
Latitude: unknown
Longitude: unknown
Processes Log
Progress Elapsed Est. Remaining Pid Tool Host Status
1.93s 0.00s 5573 smbenum (... 192.168.0.1 Finished
2.52s 0.00s 5608 smbenum (... 192.168.0.1 Finished
33.30s 66.70s 5578 snmp-defa... 192.168.0.1 Running
23.32s 76.68s 5607 nmap (stag... 192.168.0.1 Running