OSCP like Buffer Overflow - PART 1

Introduzione

Un buffer overflow si verifica quando i dati in eccesso sovrascrivono la memoria adiacente, causando crash o esecuzione di codice. In OSCP, i candidati devono identificare, analizzare e sfruttare manualmente tali vulnerabilità.

Passaggi chiave in OSCP

- Fuzzing: identificare l'input vulnerabile.
- Controllo EIP: sovrascrivere il flusso di esecuzione.
- Generare Shellcode: utilizzare strumenti come Mona.py.
- Bypass delle protezioni: gestire DEP, ASLR, NX.

Strumenti utilizzati

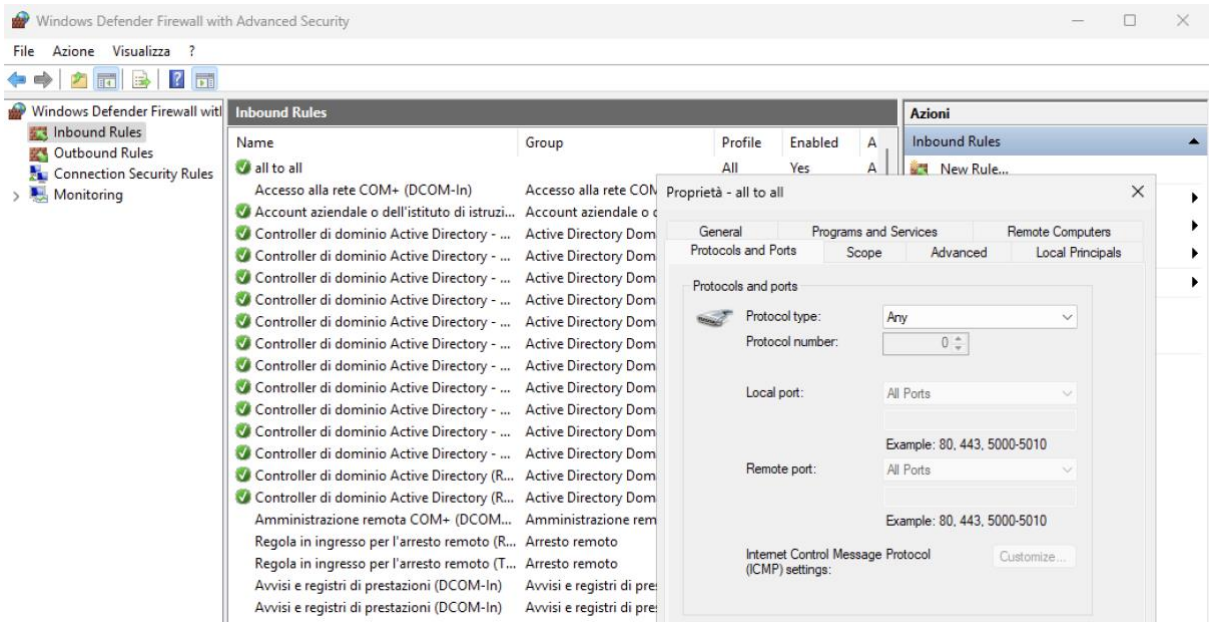
- Fuzzing: Boofuzz, Spike
- Debug: Immunity Debugger, GDB
- Exploit Dev: Python, Metasploit

Dettagli tecnici e analisi

Per questo esercizio stiamo usando il file oscp.exe installato nella macchina virtuale Windows Metasploitable basata su architettura x86. Inoltre, dobbiamo usare Immunity Debugger e, nel caso in cui mona.py non sia installato di default, dobbiamo installarlo manualmente e metterlo nella cartella dei file python del Debugger. La VM dell'attaccante è Kali Linux.

Nel mio caso ho assegnato l'indirizzo IP 192.168.101.3 per Kali Linux e 192.168.101.2 per la macchina Windows della vittima. Il ping della macchina Windows fallisce, inoltre nmap mostra che la porta 1337 (oscp.exe) è filtrata. Ciò indica che potrebbe esserci un firewall che limita il traffico:

Per procedere al passaggio successivo, sto semplicemente creando 2 regole del firewall per l'ingresso e l'uscita che consentono tutti i tipi di traffico:



Per connettermi al computer di destinazione, utilizzo lo strumento netcad e poi scrivo HELP come indicato nelle istruzioni:

```
┌──(rinatrustamov㉿kali)-[~]
└─$ nc -nv 192.168.101.2 1337
(UNKNOWN) [192.168.101.2] 1337 (?) open
Welcome to OSCP Vulnerable Server! Enter HELP for help.
HELP
Valid Commands:
HELP
OVERFLOW1 [value]
OVERFLOW2 [value]
OVERFLOW3 [value]
OVERFLOW4 [value]
OVERFLOW5 [value]
OVERFLOW6 [value]
OVERFLOW7 [value]
OVERFLOW8 [value]
OVERFLOW9 [value]
OVERFLOW10 [value]
EXIT
OVERFLOW1
UNKNOWN COMMAND
```

Per determinare quale comando OVERFLOWX è vulnerabile, dobbiamo effettuare il fuzz dell'applicazione inviando input sempre più lunghi a ciascun comando e osservare quale fa crashare l'applicazione. Per questo ho utilizzato uno script python personalizzato. Allo stesso tempo, per mostrare e capire meglio, ho aperto il task manager in Windows. Capiremo che la macchina si è bloccata quando il task manager elimina oxcp.exe e/o quando il programma python smette di inviare byte. Possiamo vederlo nel seguente video.

Ora dobbiamo determinare il numero esatto di byte necessari per sovrascrivere il registro EIP. Questo è fondamentale perché una volta che controlliamo EIP, possiamo reindirizzare il flusso di esecuzione al nostro shellcode. Quando si verifica un buffer overflow, i dati traboccano dal buffer nelle regioni di memoria adiacenti, incluso EIP (Extended Instruction Pointer).

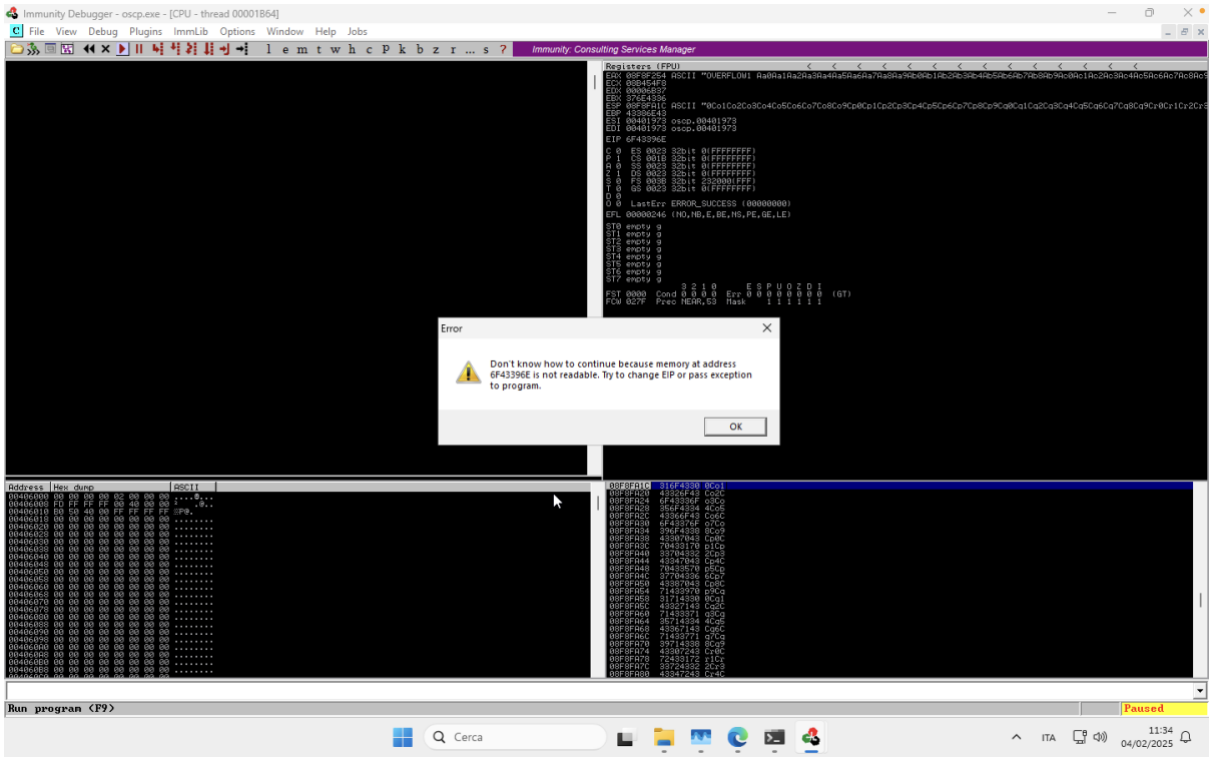
- EIP controlla la prossima istruzione da eseguire.
- Se possiamo sovrascrivere EIP con un valore controllato, possiamo reindirizzare l'esecuzione al nostro shellcode.

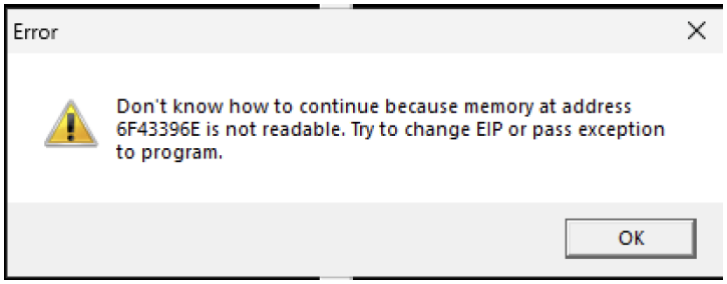
Per questo dobbiamo generare un modello univoco che è un modello da 3000 byte creato utilizzando gli strumenti Metasploit:

Poi dobbiamo inviare il pattern al target in modo che l'applicazione target (oscp.exe) si blocchi. Per questo ho usato uno script python personalizzato:

```python
exp1.py > ...
import socket

target_ip = "192.168.101.2"
target_port = 1337

# Replace with your generated pattern
buffer = b"OVERFLOW1 " + b"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((target_ip, target_port))
print(s.recv(1024))  # Receive banner
s.send(buffer + b"\r\n")
s.close()
print("[*] Sent pattern to identify offset")
```

E poi dobbiamo intercettare il crash usando Immunity Debugger in modo da poter ottenere il valore EIP. Stiamo prima collegando il servizio oscp.exe in Immunity Debugger e poi eseguendo gli script python. Questo intercetterà il valore EIP nel Debugger:





Come potete vedere, il valore è 6F43396E. Ora, per trovare l'offset, stiamo di nuovo usando gli strumenti metasploit, e ci restituiranno la corrispondenza esatta all'offset. Nel nostro caso era 1978. Ciò significa che i primi 1978 byte hanno riempito il buffer e i successivi 4 byte hanno sovrascritto EIP.

Infine, eseguiamo una prova di concetto in python che si collegherà al server vulnerabile e invierà un payload che non solo si bloccherà, ma confermerà che i nostri offset sono affidabili. Il payload che invieremo è: AAAA... (1978) ...AABBBBCCCCCCCCCCCCCCCC. Quando il programma si blocca, l'EIP dovrebbe essere BBBB (0x42424242) e ESP dovrebbe puntare a CCCCCCCCCCCCCCC.

Possiamo guardare il video qui.