REPORT ATTIVITA’ EXPLOIT BUFFER OVERFLOW

BW III

Obiettivo:

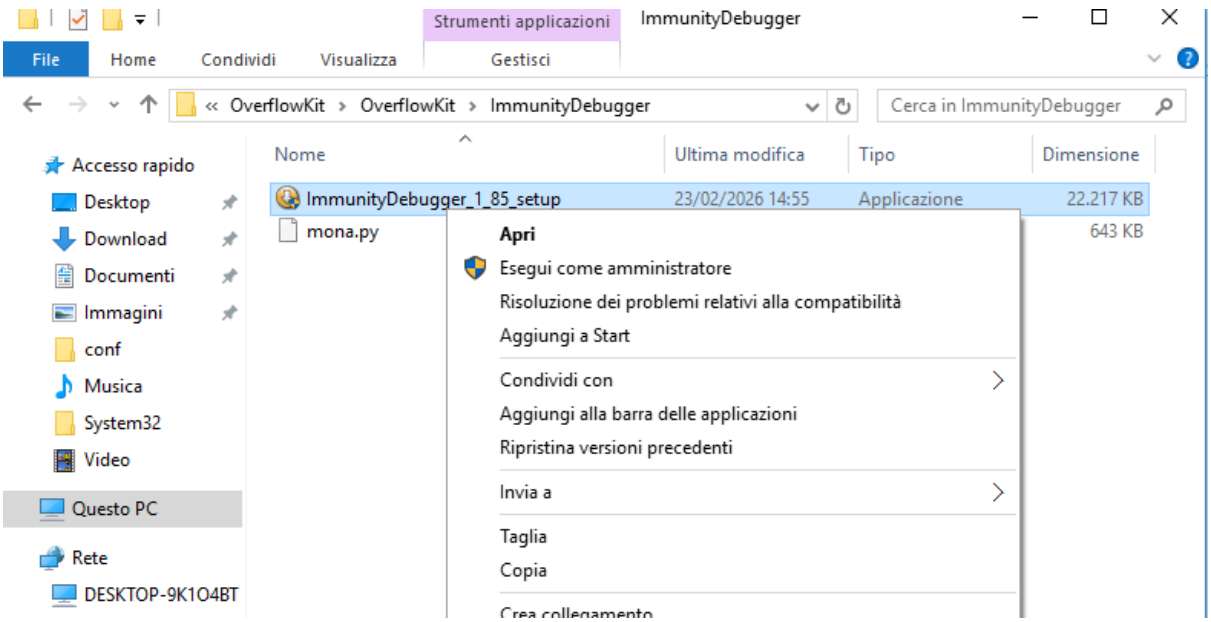
dirottare il flusso di esecuzione di un programma vulnerabile, sovrascrivendo l'indirizzo di ritorno (EIP salvato sullo stack) per far sì che, al termine della funzione, il processore non torni alla funzione chiamante ma esegua il payload malevolo.

Indice dei risultati

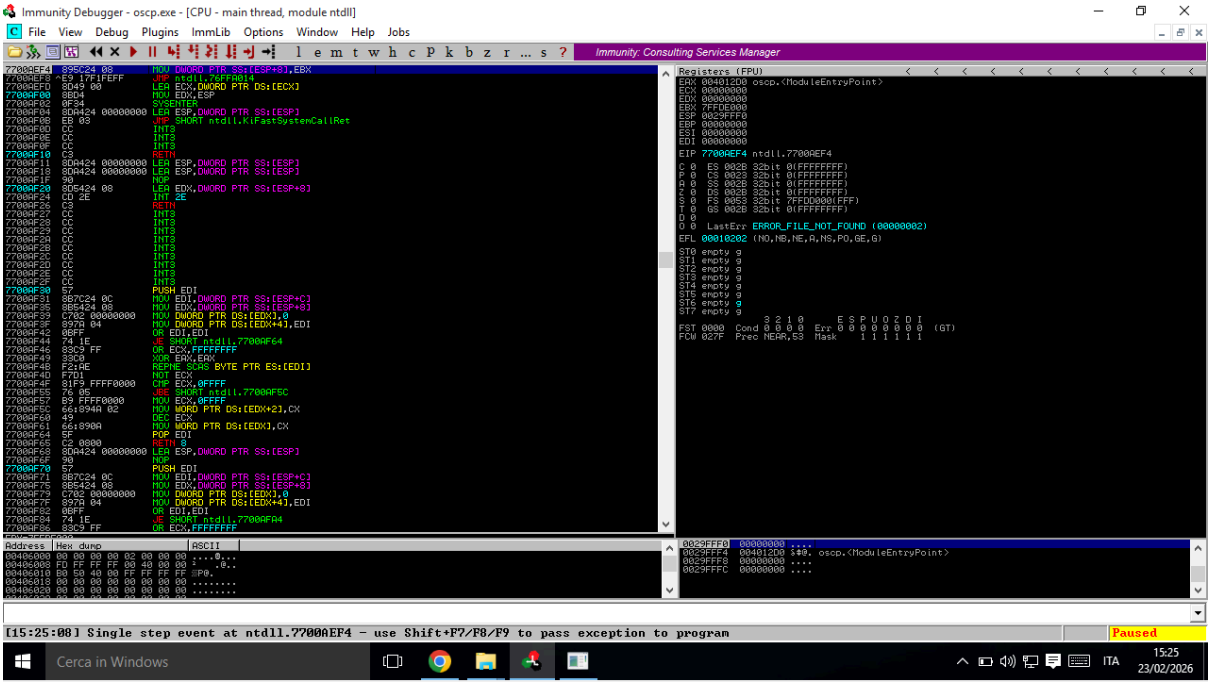
- **Identificazione della Vulnerabilità**: È stata determinata la capacità del buffer di ricevere input arbitrari, identificando l'offset esatto a **1978 byte** per il controllo del registro EIP.
- **Analisi della Memoria**: Grazie all'utilizzo di **Mona.py** in ambiente Immunity Debugger, è stato individuato un indirizzo di ritorno stabile (`0x625011af`) all'interno della libreria `essfunc.dll`, privo di protezioni ASLR/DEP.
- **Sviluppo del Payload**: Lo shellcode è stato ottimizzato escludendo i caratteri non validi (BadChars) identificati (`\x00\x07\x2e\xa0`), garantendo la corretta esecuzione del codice malevolo.
- **Compromissione del Sistema**: L'invio del payload finale, supportato da una rampa di NOP per la stabilità, ha permesso di stabilire una Reverse Shell verso la macchina attaccante sulla porta 1337.

Avvio ambiente e debugger

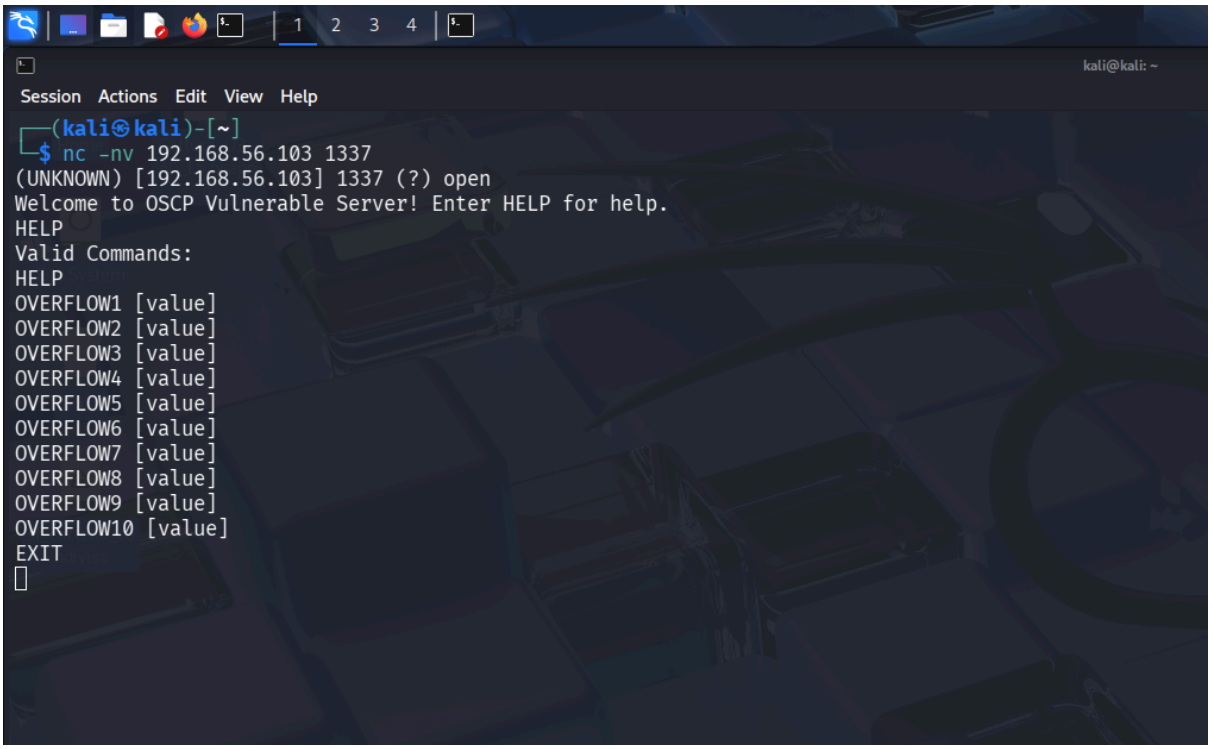
dopo il download Il programma vulnerabile **oscp.exe** è stato caricato all'interno di Immunity Debugger sulla macchina Windows target.
Il debugger è stato avviato con privilegi amministrativi per garantire il corretto funzionamento degli strumenti di analisi.



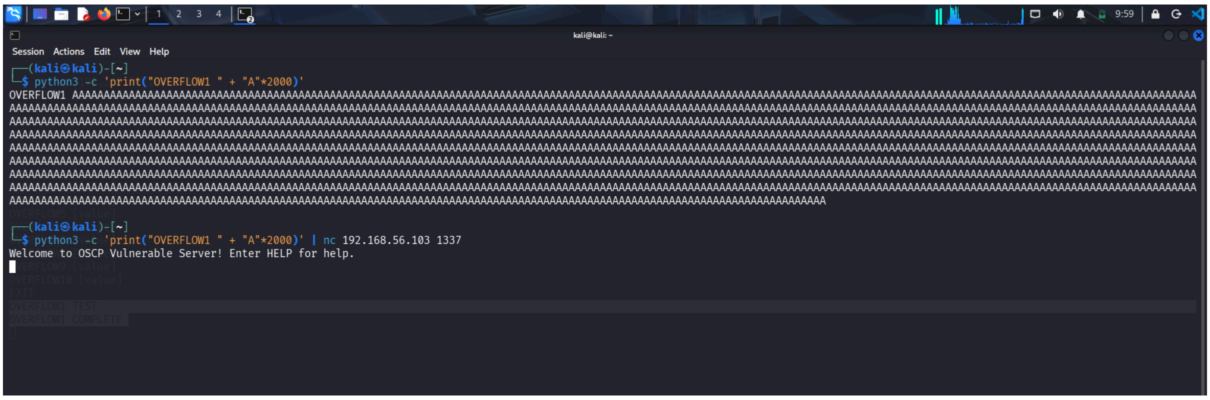
Successivamente è stato eseguito il programma, permettendo al servizio di avviarsi e rimanere in esecuzione sotto il controllo del debugger.

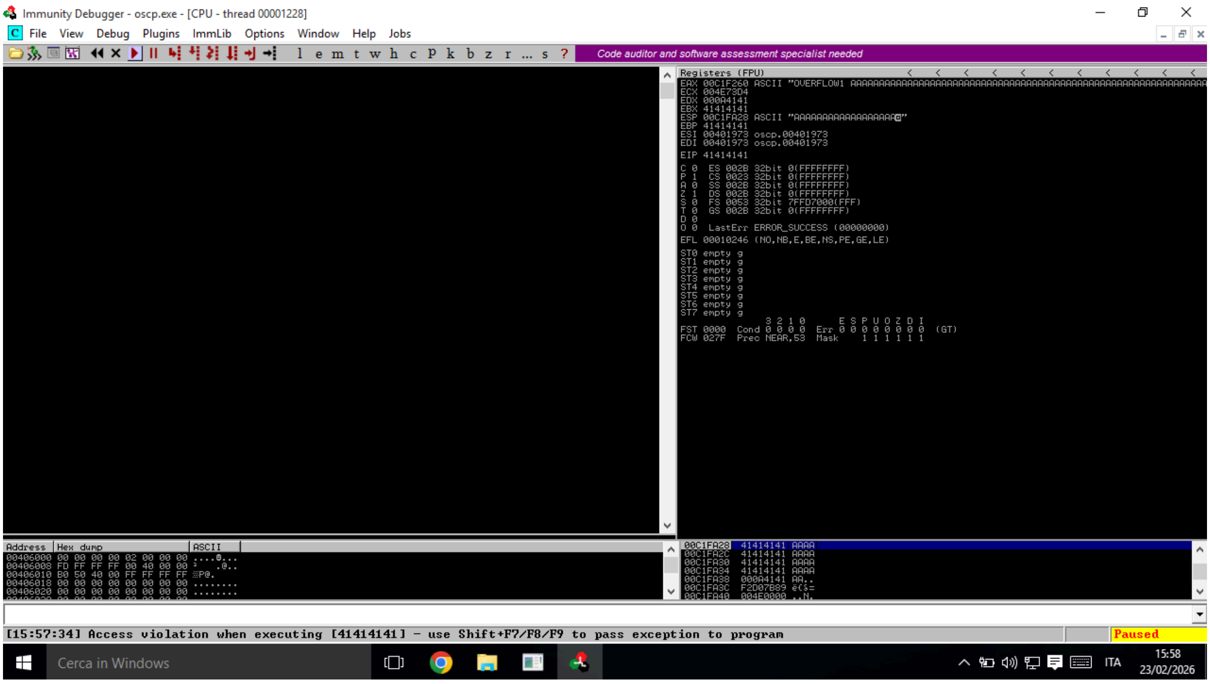


La disponibilità del servizio vulnerabile è stata verificata dalla macchina Kali utilizzando il tool netcat. La connessione alla porta TCP 1337 del sistema target è risultata positiva, confermando che il programma oscp.exe è in esecuzione e in ascolto. I comandi OVERFLOW1–10 suggeriscono più punti di overflow

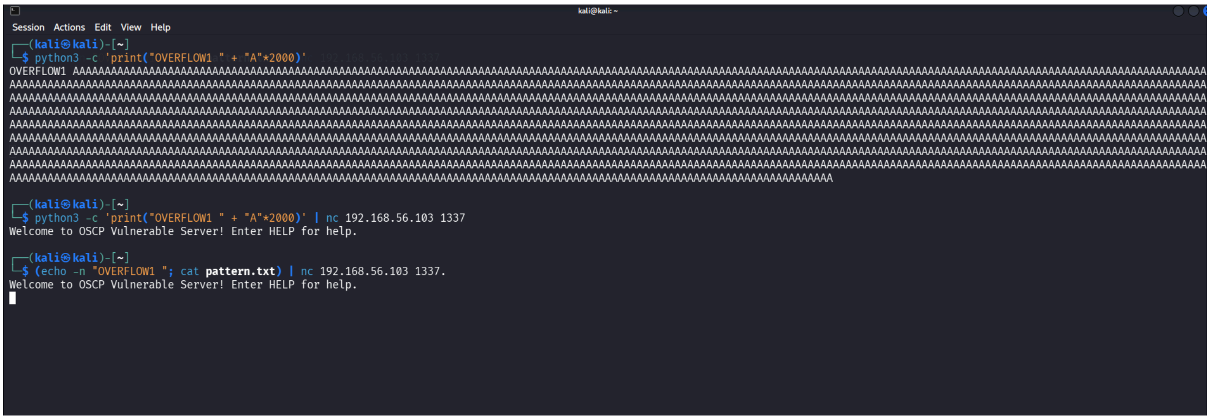


L'invio di un input eccessivamente lungo ha causato il crash dell'applicazione e la sovrascrittura del registro EIP con il valore 41414141, confermando la presenza di una vulnerabilità di Buffer Overflow controllabile.

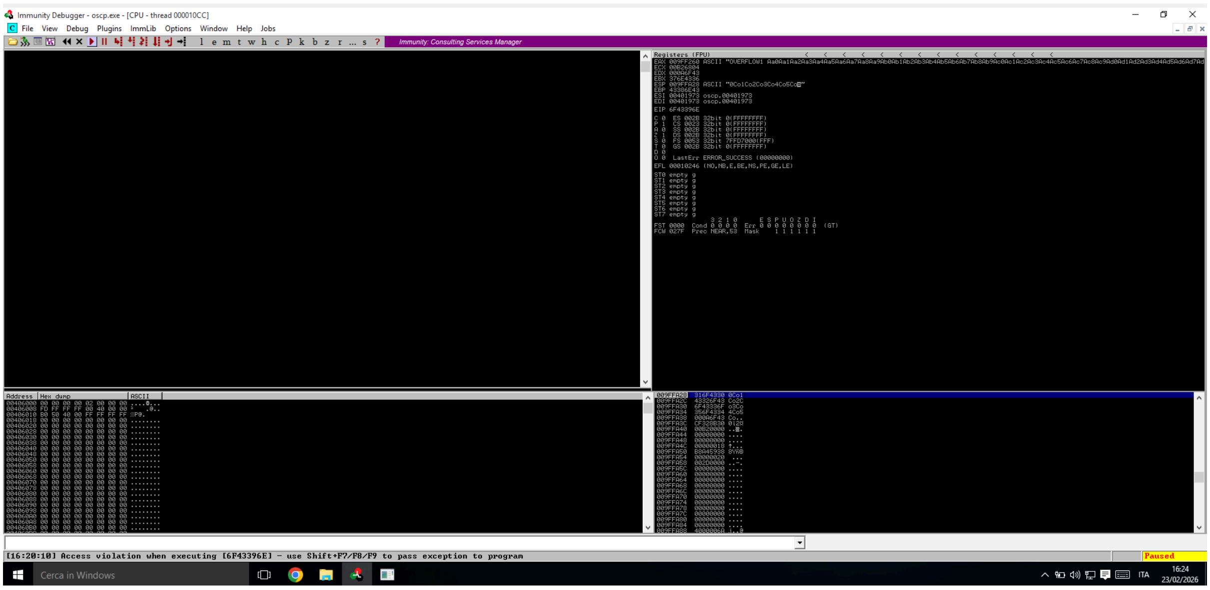




Durante la fase di analisi della vulnerabilità, è stato inviato al servizio il comando **OVERFLOW1** accompagnato da un pattern non ripetitivo generato tramite lo strumento `pattern_create` del framework Metasploit. L'obiettivo era individuare con precisione la posizione del registro EIP all'interno del buffer.

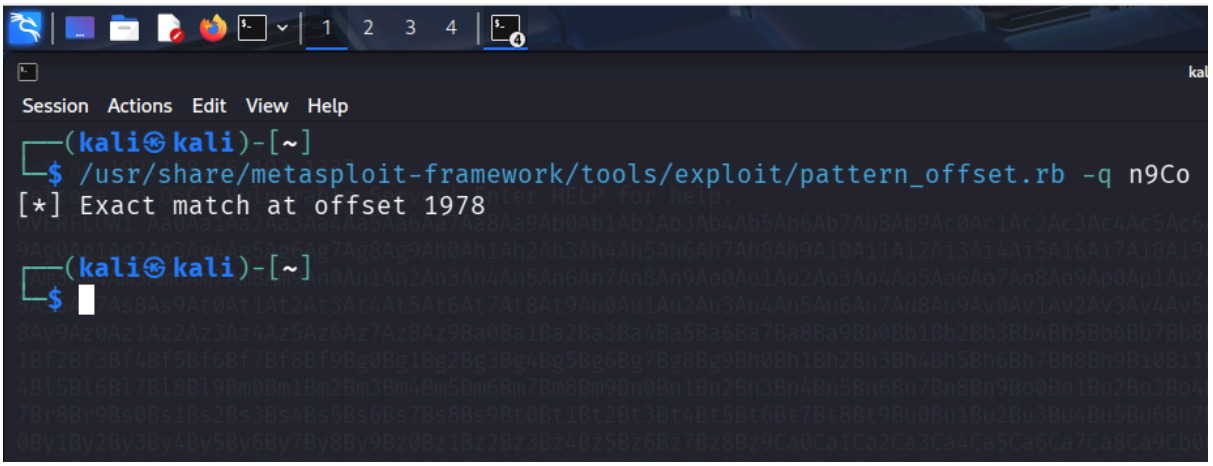


A seguito dell'invio del payload, l'applicazione è andata in crash sotto il controllo di Immunity Debugger, generando un'eccezione di tipo *Access Violation*. L'analisi dei registri ha evidenziato la sovrascrittura del registro EIP con il valore esadecimale **0x6F43396E**.



EIP = 6F43396E | Serve a sapere dopo quanti byte lo sovrascrivi

Utilizzando il pattern ciclico generato con lo strumento `pattern_create`, è stato possibile determinare con precisione l'offset del registro EIP tramite `pattern_offset`. L'analisi ha restituito un offset pari a **1978 byte**, ovvero la posizione esatta all'interno del buffer in cui avviene la sovrascrittura del puntatore di istruzione.



EIP viene sovrascritto dopo 1978 byte

Successivamente, è stato costruito un payload di verifica composto da:

- una sequenza di caratteri "A" lunga 1978 byte
- la sequenza "BBBB" per sovrascrivere EIP
- una sequenza di caratteri "C" per verificare il contenuto puntato da ESP

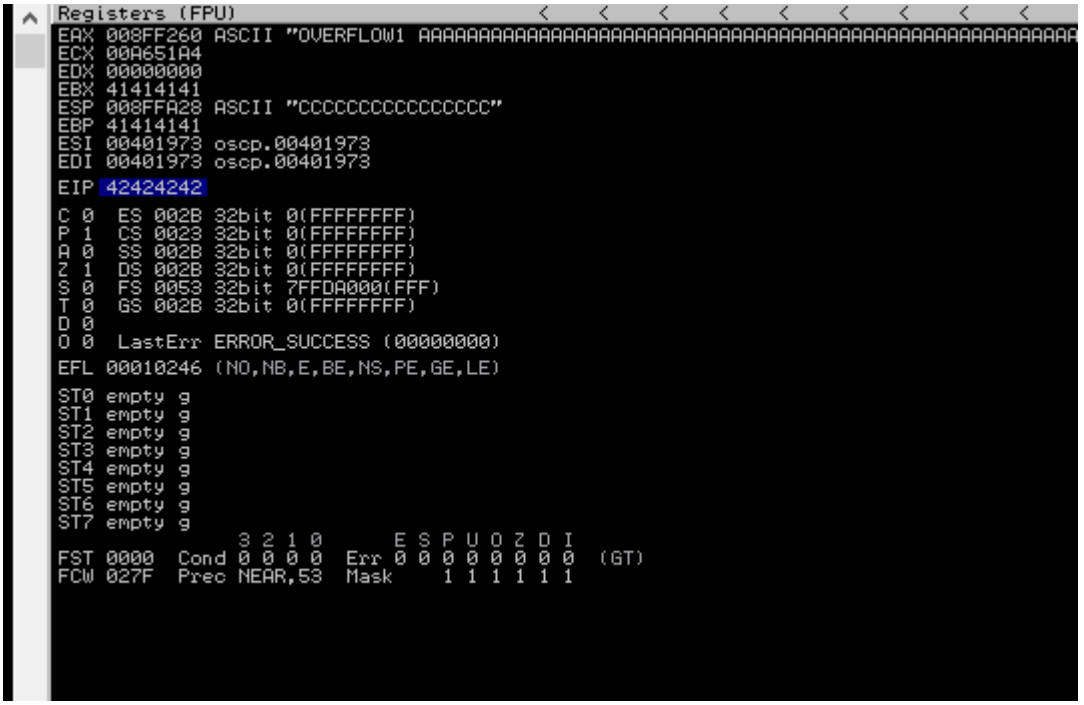
```
┌──(kali㉿kali)-[~]
└─$ nano poc.py

┌──(kali㉿kali)-[~]
└─$ python3 poc.py
Traceback (most recent call last):
  File "/home/kali/poc.py", line 19, in <module>
    s.recv(1024)
    ~~~~~~^^^^^^
TimeoutError: timed out

┌──(kali㉿kali)-[~]
└─$
```

Non c'è risposta dal server perché è crashato (timeout)

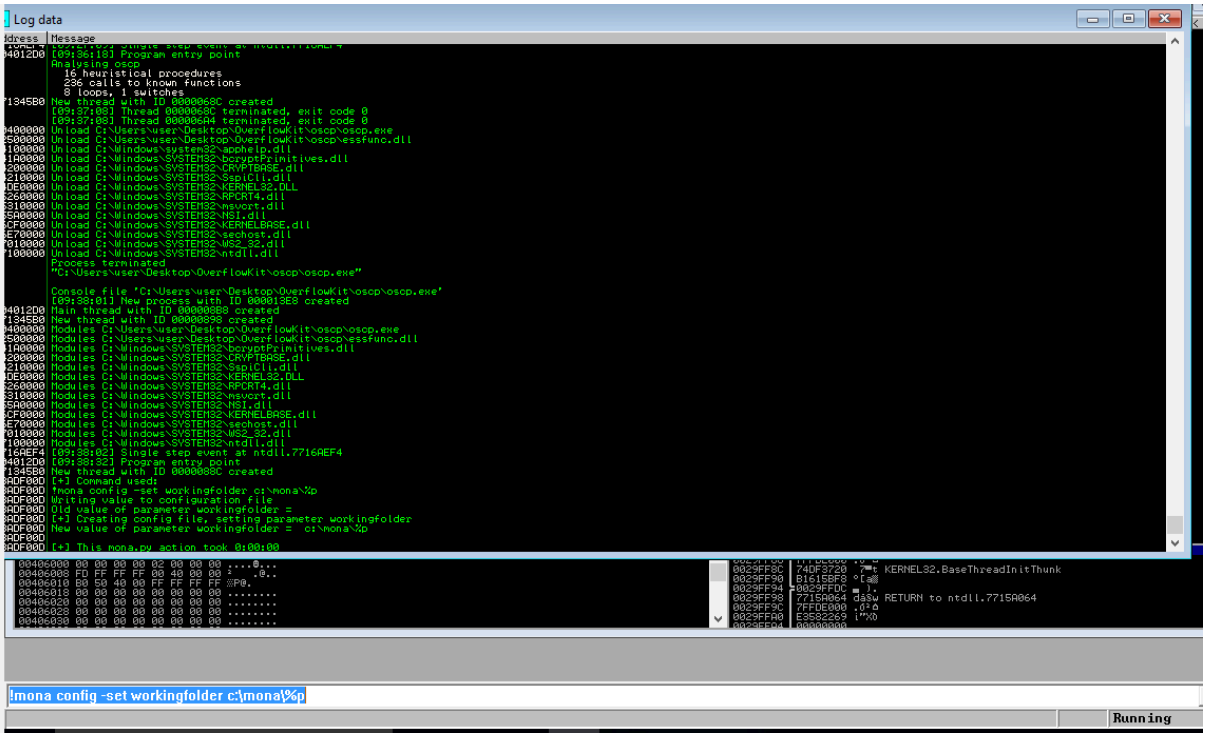
L’invio del payload ha prodotto un nuovo crash, durante il quale il registro EIP conteneva il valore **0x42424242** (“BBBB” in ASCII), confermando il pieno controllo sul flusso di esecuzione. Inoltre, il registro ESP puntava all’inizio della sequenza di caratteri “C”, dimostrando che lo stack conteneva il payload controllato dall’attaccante.



BADCHARS

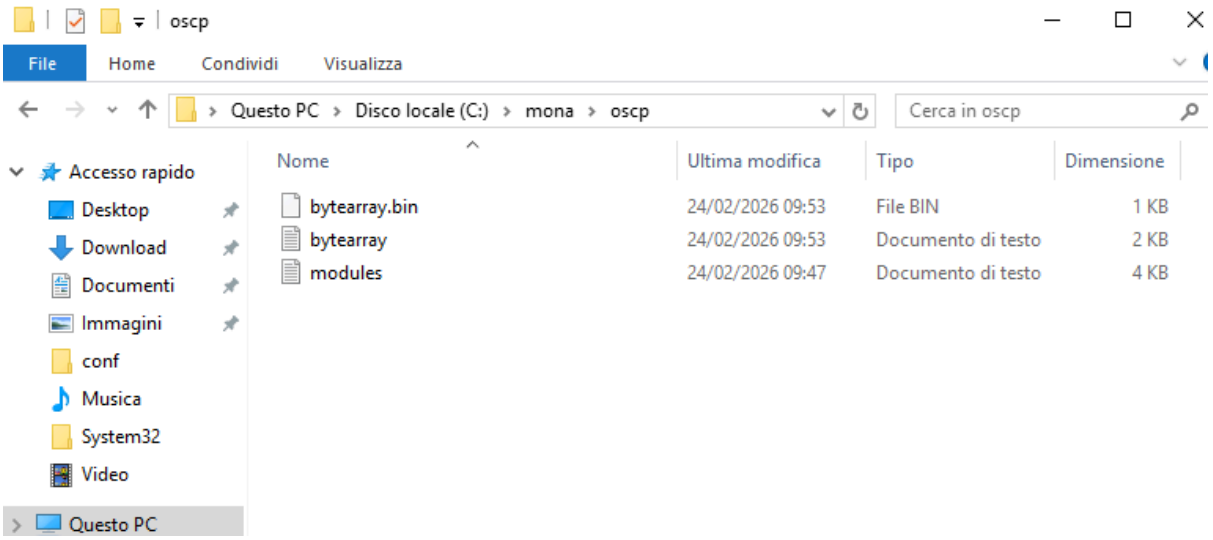
È stato configurato il plugin Mona all'interno di Immunity Debugger per facilitare l'analisi della lista dei caratteri che il programma non riesce a gestire (badchars)

Tramite il comando: !mona config -set workingfolder c:\mona\%p

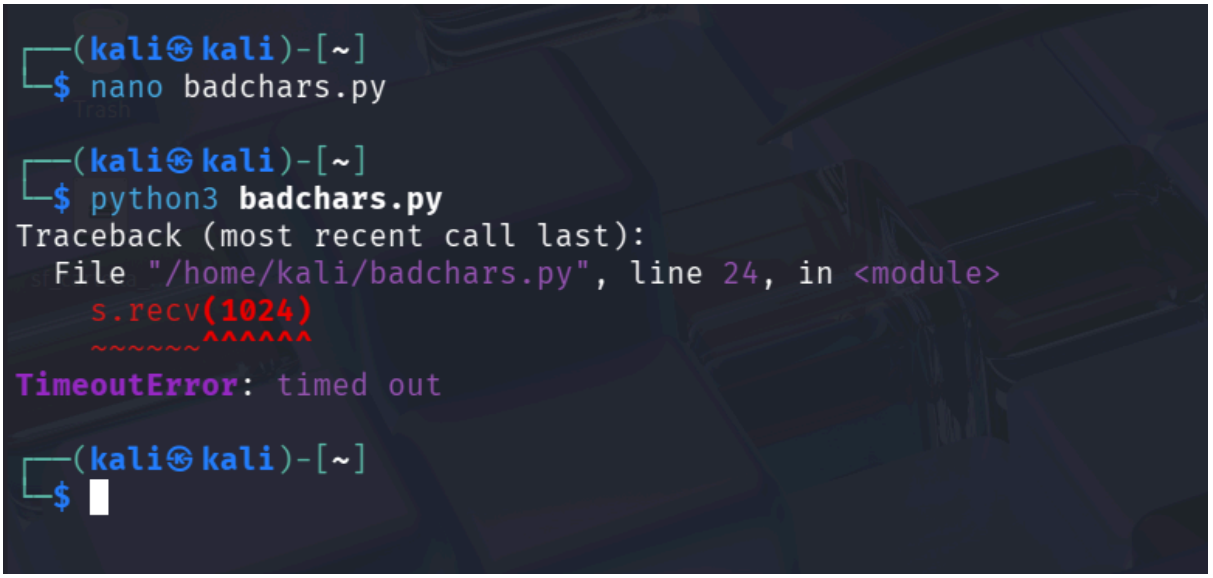


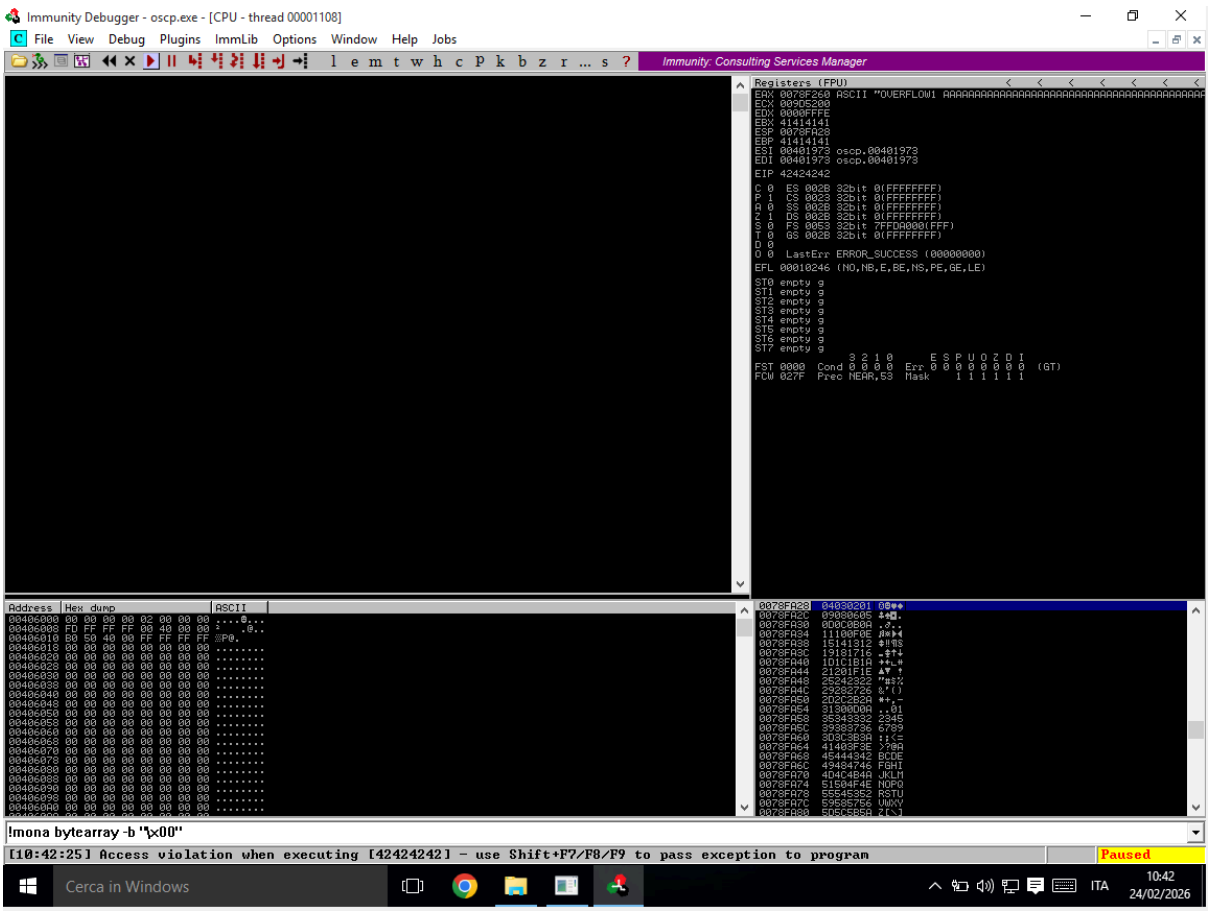
È stato utilizzato il plugin per generare un bytearray di riferimento contenente tutti i possibili valori da 0x00 a 0xFF, escludendo il null byte (0x00), tramite il comando: !mona bytearray -b "\x00"

Il bytearray prodotto è stato salvato nella directory di lavoro configurata (`C:\mona\oscp\`) nei file `bytearray.txt` e `bytearray.bin`.

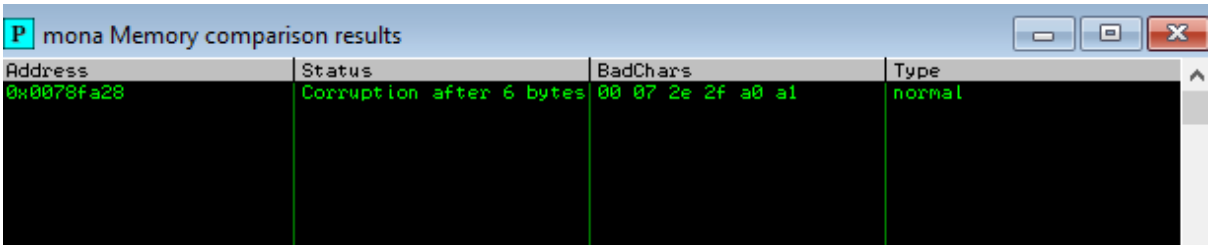


Dopo l'invio del payload contenente il bytearray, il programma è andato in crash sotto Immunity Debugger. a mona abbiamo chiesto di generare un bytearray di riferimento con tutti i valori possibili ma escludendo i byte specificati come "badchar noti"



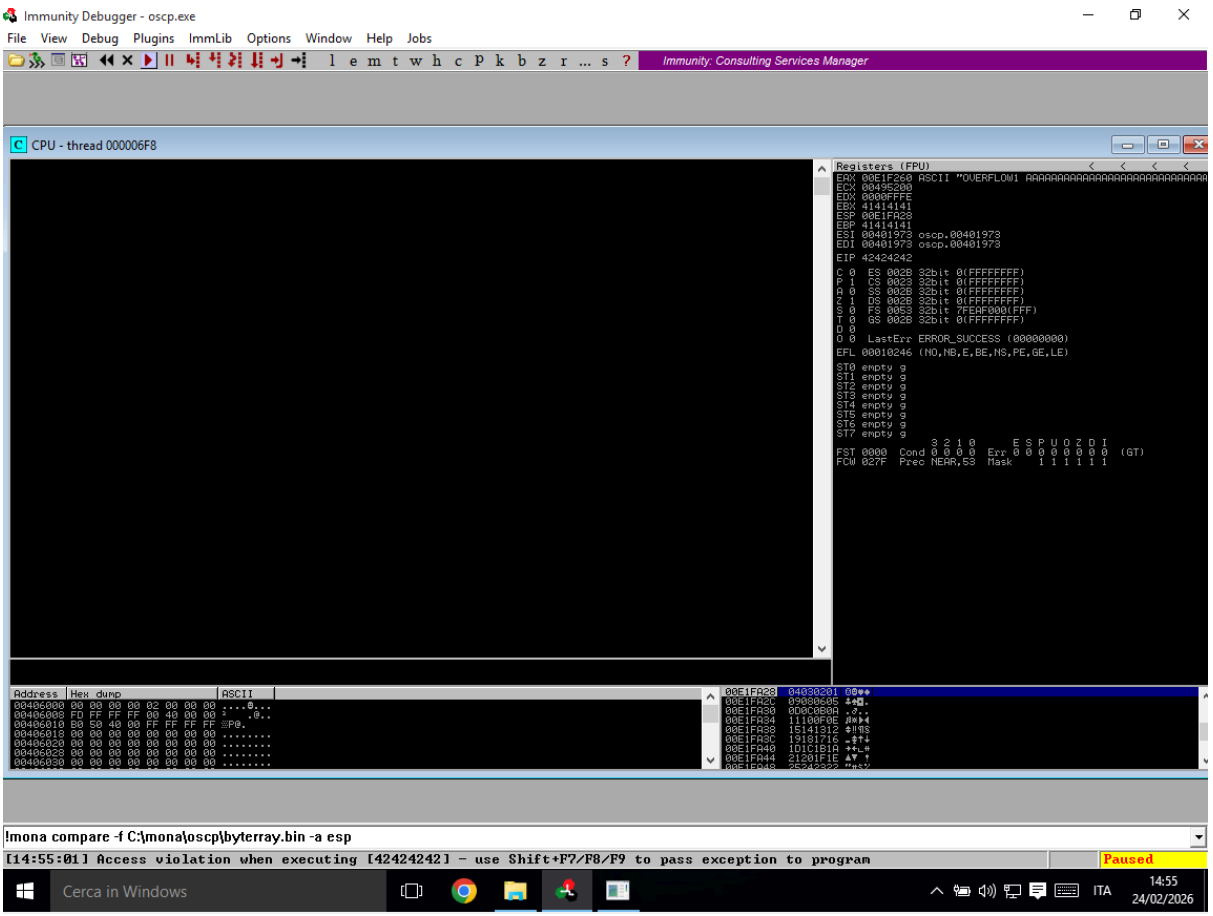


È stato eseguito il confronto tra il bytearray di riferimento e i dati presenti nello stack all'indirizzo puntato da ESP mediante il comando `!mona compare`. Il confronto tra il bytearray di riferimento e i dati presenti nello stack ha evidenziato una corruzione a partire dal settimo byte, indicando possibili bad characters



Il payload aggiornato è stato nuovamente inviato al servizio vulnerabile, causando un nuovo crash controllato dell'applicazione. La ripetizione del confronto ha permesso di verificare la corretta esclusione del carattere individuato e l'assenza di ulteriori corruzioni nei byte iniziali. Questo processo iterativo consente di determinare

con precisione l’insieme dei bad characters reali, requisito fondamentale per la costruzione di uno shellcode funzionante.

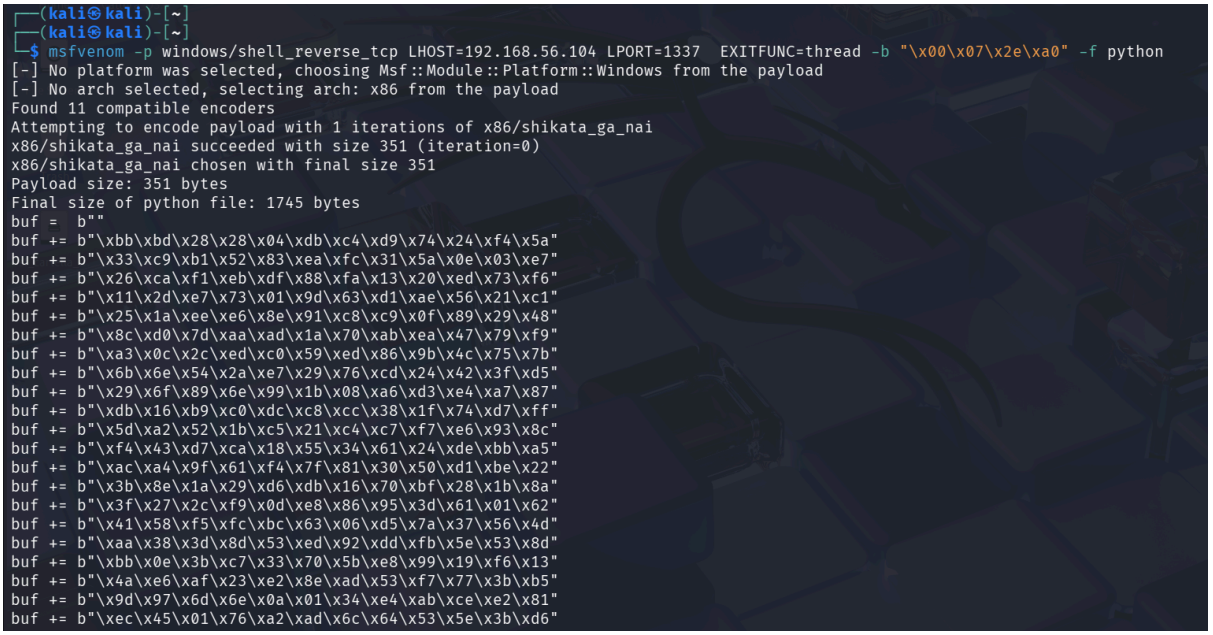


GENERAZIONE SHELLCODE PER OTTENERE RCE

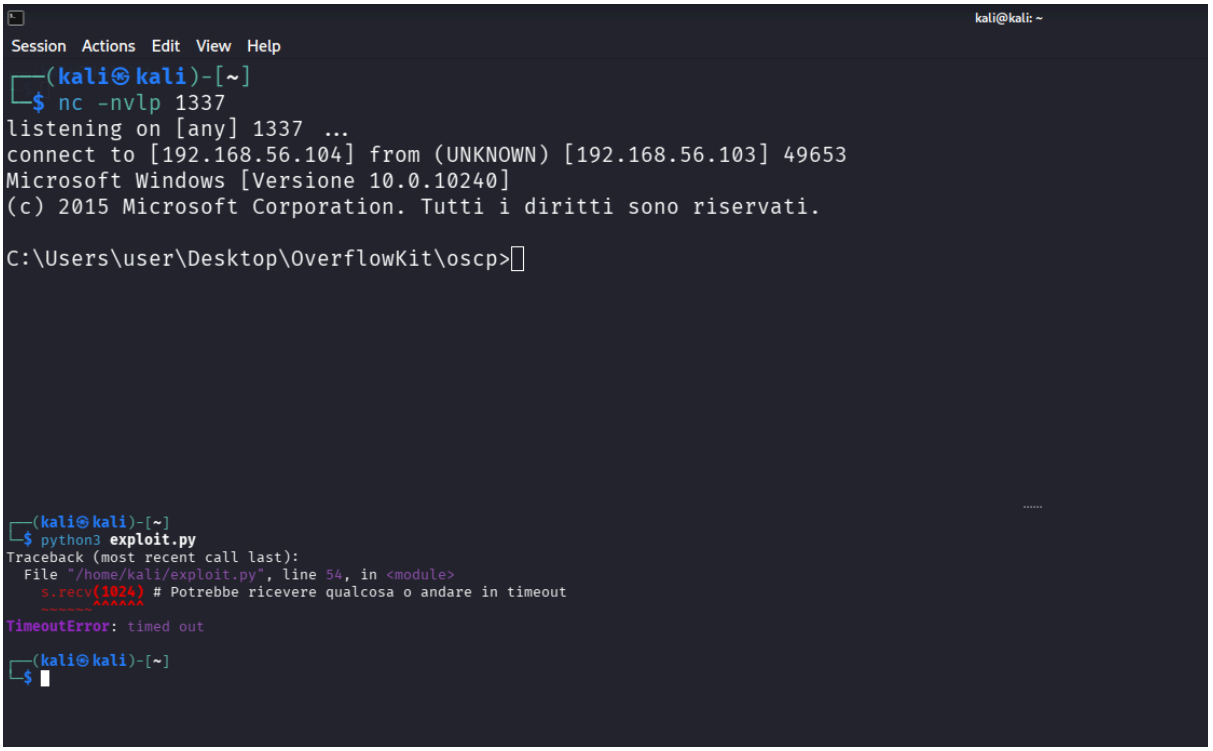
l'obiettivo è trovare un'istruzione già esistente all'interno del programma (o di una sua libreria) che permetta di "instradare" il processore verso il tuo codice malevolo.

Per reindirizzare il flusso di esecuzione verso lo shellcode caricato nello stack, è stata effettuata la ricerca di un'istruzione `JMP ESP` (gadget) all'interno dei moduli caricati dal programma. Utilizzando il comando `!mona jmp -r esp -cpb "\x00\x07\x2e\xa0`" sono stati filtrati i risultati per escludere gli indirizzi contenenti i **BadChars** precedentemente identificati.

Per la fase conclusiva, è stato generato uno shellcode `windows/shell_reverse_tcp` tramite **msfvenom**, configurato con l'IP 192.168.56.104 della macchina attaccante (LHOST) e la porta di ascolto (LPORT 1337)



L'exploit finale è stato lanciato dopo aver configurato un listener sulla macchina attaccante (Kali Linux) tramite l'utility Netcat sulla porta **1337**. Lo script Python ha generato un `TimeoutError` in fase di ricezione del banner (dovuta all'immediato crash del servizio legittimo a favore dello shellcode), l'attacco è andato a buon fine.



Conclusione

L'attività di analisi sulla vulnerabilità del servizio oscp.exe ha dimostrato con successo la presenza di una falla di tipo Buffer Overflow nel comando `OVERFLOW1`. Attraverso un processo metodico di exploit development, è stato possibile trasformare un semplice crash del programma in una Remote Code Execution (RCE) completa.
