Buffer Overflow Exploitation

Exploit Development tutorial, reverseengineering

Sk0xic

3 Oct '17

Hi 0x00ers.

This is my first post and my goal in it is to share and detail how you can exploit a buffer overflow by doing a detailed analysis of the executable and for that I will solve a challenge proposed by ricnar in its reversing course, it is clear that i just started on the subject of reversing and if I am wrong in something I ask you heartily that let me know

Author Assigned Level: Newbie

Community Assigned Level:

- Newbie
- Wannabe
- Hacker
- Wizard
- Guru

65 voters

Show results

Required Skills

- C++ language, a basic level would be fine
- x86 Intel Assembly
- Basic IDA Pro or Free usage

#The Binary

Mediafire 46
Virustotal 32

Buffer Overflow

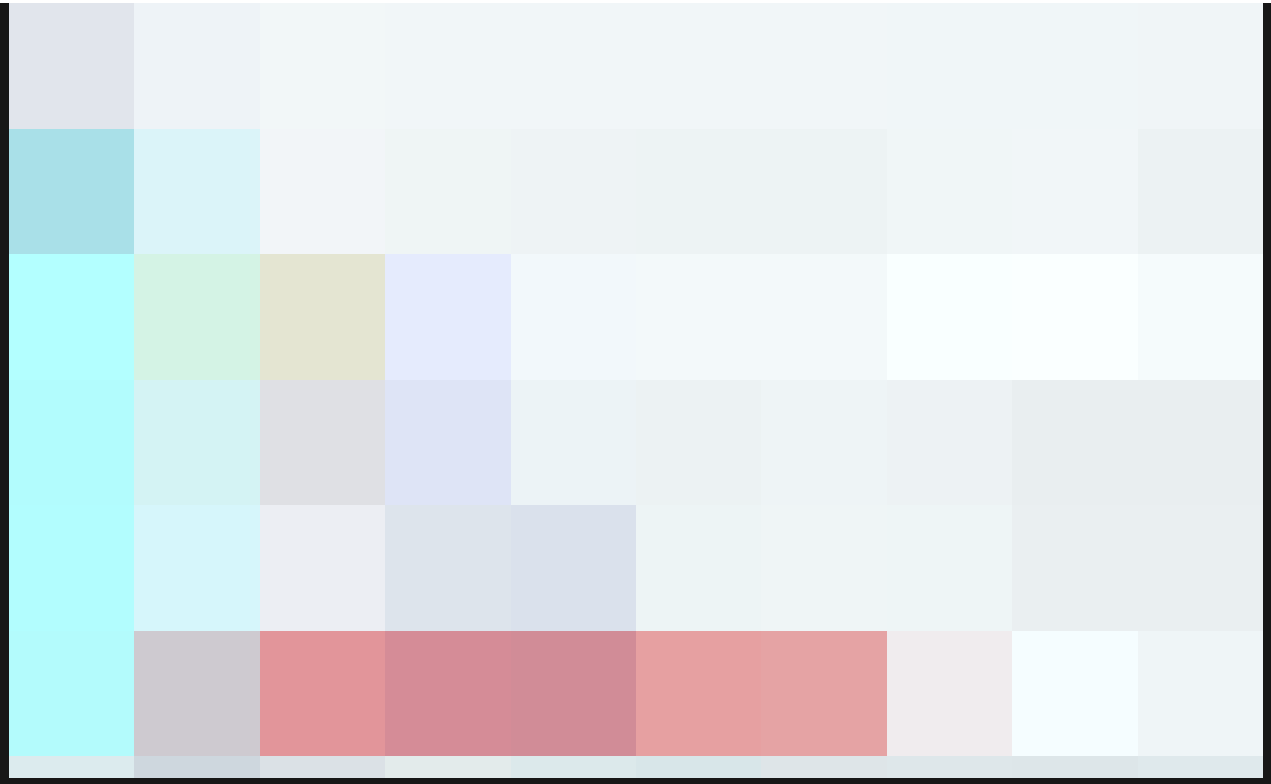
Buffer overflow occurs when a program reserves a memory zone or buffer to store data and for some reason the size of the data to be copied is not properly checked and the buffer is overflowed by copying more than the reserved size being able to step variables, arguments and pointers which are in the memory.

The simplest type of buffer overflow is the stack overflow, which is when there is an overflow in a reserved buffer in the stack and is the one that i will explain how to exploit

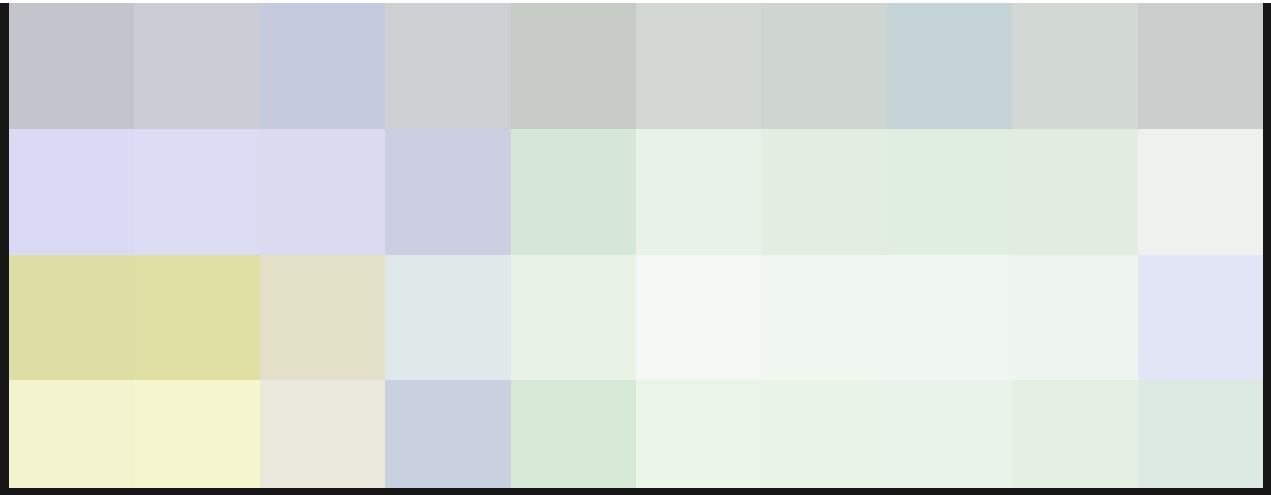
Let’s go

First let’s see what we have in the executable, so we run it.

We can see that he asks us to enter a number, I typed 1337 and the program closed. Then we do not know what it is that does, let’s analyze it with IDA

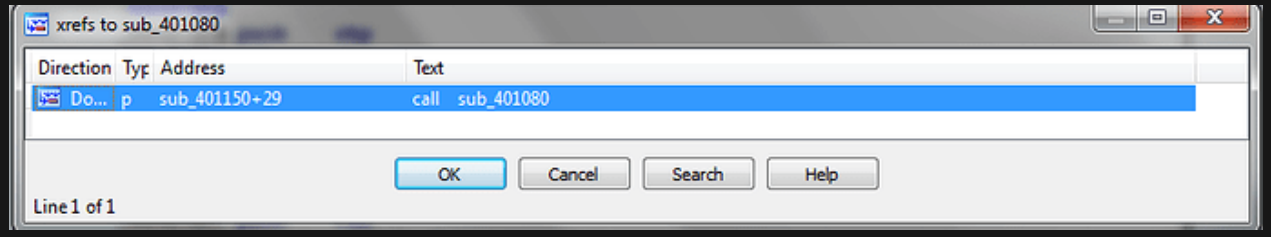


IDA leaves us in the entry point, but we seen in the executable a string “Please Enter Your Number of Choice:”. Then we will look for it, View-Open subviews-Strings, there we see many strings, Ctrl + F and our string



We double click on it, then we see where it is referenced by clicking on it and pressing the X key and ok

If we follow the calls “sub_4011B0” and “sub_4011F0” we will realize that it is a printf and a scanf, respectively, so we can rename it. Then here we see something that is possibly a structure because when you pass as an argument an address and then it is retrieved and added offsets to access the fields in each place that is used, it is possibly the direction of a structure. Let’s see the references of this function.



I see there’s a call, I go there

I see that the argument is an address, which agrees with the idea of structure, so we create one, without knowing the size, without knowing the fields or anything, we will reverse them little by little.

We see that the maximum offset that I find so far is 0x14, so I will create a structure of that length, if it becomes bigger I will enlarge it. View-Subviews-Structures then Edit-Add struct type

There it was created called “MyStruct” with size 0, now I will do a trick for when I still do not know the fields or anything and I want to give a size, first press D on the word “ends”, to add a single field.

There I add a field of 1 byte long DB, if I would press D it would change every time to word DW and then to DWORD DD.
But here as we do not know, we leave it like this and we right click on the structure to expand it since I have seen a field in 0x14, so as to fill that field with a dword, it needs 4 more bytes, I'll create it from 0x18, I'll will add 0x17 to the byte it had.

I see that I stay with size 0x18 for now we will leave it like this, if we need more we enlarge it. As we saw that arg_0 is the argument that corresponds to a structure, we can rename it to _struct

If we decompile the function with F5 we see that it is not right

I see that the type of variable is int and not that of a structure as it seems, let’s change that. Right click on the argument, convert to struct and we choose the one we create

Obviously Buf is the structure and there gets its address and passes as an argument, let's see Buf in the representation of the stack. As the structure does not need to be created because it already exists, I just have to say that Buf is MyStruct type, for that ALT + Q in Buf.

```
-00000020 ;
-00000020
-00000020 Buf              MyStruct ?
-00000008 var_8            dd ?
-00000004 var_4            dd ?
+00000000  s               db 4 dup(?)
+00000004  r               db 4 dup(?)
+00000008
+00000008 ; end of stack variables
```

Rename Buff to Struct and return to the function

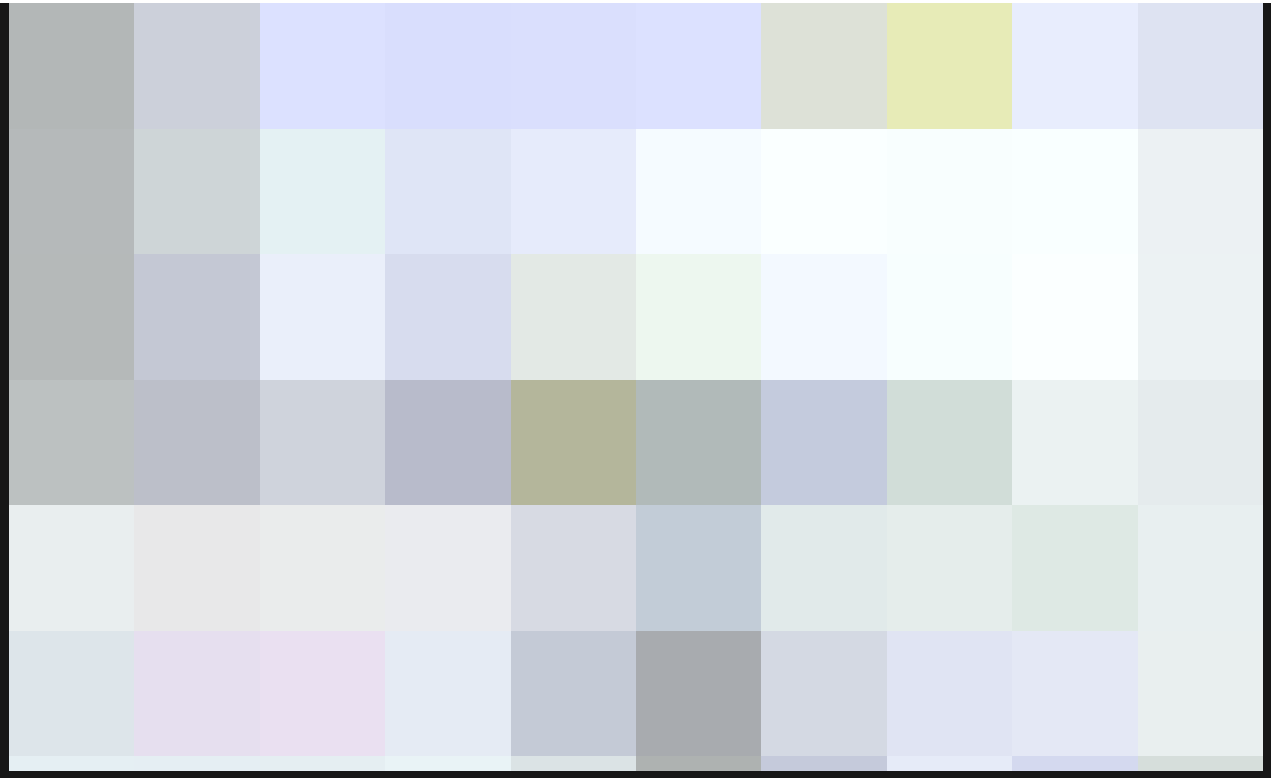
We see that the field in 0x10 is a dword where it receives the value of scanf, so we go to MyStruct and in 0x10 we press the D until it is of type DWORD DD and we change it to number.

The other entry is the field 0x14 that is used in the loop to remove the 0A I will name it c.

We press T on it and choose the field that corresponds in the structure

Rename the function to “enter” and we continue to reverse the following function

It also passes as argument the structure (buf), but as we see that it is char type we change it, f5-convert to struct * and we change the name with T to the corresponding fields



In the function we see a comparison between the number we enter and 0x10, then comes a jle that tells us that if the number is less or equal, considering the sign, jumps to “loc_401024” and if it does not come out. That explains why when I typed 1337 it came out

Then it uses as size of get_s the number that we enter, and the other argument must be a buffer that is at the beginning of the structure because it uses the start address of the same, so I go to MyStruct and in 0x0 i press D once to create a single-byte field.

Then right click on it, array, the length of the buffer will be 16 i accept it and rename it to buffer

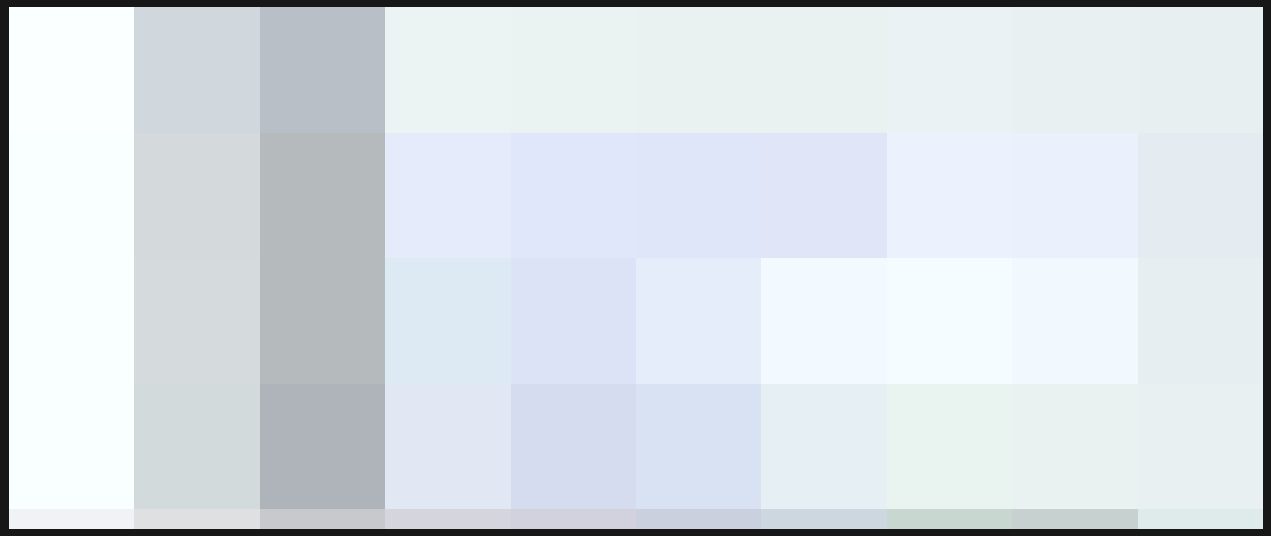
The issue is that with gets_s the buffer may be overfloded, since the check passes negative values that when used as size, will be taken as unsigned values, and will be large, If for example we pass 0xffffffff in the comparison will be -1 because it is signed and it will be less than 0x10, but using it as size will be the positive value 0xffffffff which allows us to pass the number of characters we want in the gets_s to the buffer and overflode it .

So we could rename the function as check or get whatever we want it to be representative of what the function does, Let’s see the following function.

The argument is the same so I repeat the procedure, press F5 and change the argument type, then we see that there is one more field since it is trying to compare [EAX + 0x18], which we have not defined, because the last field of MyStruct is 0x14, so we will add it and i will rename it to key

```
00000000 MyStruct        struc ; (sizeof=0x1C, mappedto_28)
00000000 buffer          db 16 dup(?)
00000010 numero          dd ?                    ; XREF: che
00000014 c               dd ?                    ; XREF: ent
00000018 key             dd ?
0000001C MyStruct        ends
```

Let’s go back to the function



At this point we know that to get the message “You are a winner man” must be MyStruct.key equal to 0x45934215, so we already know that we must surpass (I did not find a better translation)

Let’s look at the distribution of the main stack.

Obviously everything is inside Struct, the buffer and the key, so let's go to structures to see the sizes of each.

```
00000000 MyStruct        struc ; (sizeof=0x1C, mappedto_28)
00000000 buffer          db 16 dup(?)
00000010 numero          dd ?                    ; XREF: che
00000014 c               dd ?                    ; XREF: ent
00000018 key             dd ?
0000001C MyStruct        ends
```

So, we have to fill the buffer with 16 aes, then 2 more dwords and then the key, it would be something like this

> "A" * 16 + numero + c + key

A script to exploit it might look like this:

```
from subprocess import *
import struct
p = Popen([r'ConsoleApplication4.exe', 'f'], stdout=PIPE, stdin=PIPE, stderr

enter="-1\n"
p.stdin.write(enter)


numero=struct.pack("<L",0x34333231)
c=struct.pack("<L",0x90909090)
key=struct.pack("<L",0x45934215)

payload = "A" * 16 + numero + c + key + "\n"
p.stdin.write(payload)


testresult = p.communicate()[0]
print(testresult)
```

We see that it happens -1 as number to pass the check when it compares with sign against 0x10 and then the 16 bytes to fill the buffer, then the number to which I passed a correct value of 0x34333231 because overflodear will change it, then c which can be any value and then the key 0x45934215.

19

created	last reply	8	7.1k	7	30	2
Oct '17	Dec '17	replies	views	users	likes	links

Noswis Oct '17

Nice read man, It's really cool to see writeups like this! It's nice and visual and I learend a lot more about the memory representation of structs. I like reversing alot but I never really got started with exploitiation part. Stuff like this makes me want to learn more about it!

1

Valentine Oct '17

Finally!!! Somebody wrote a post on buffer overflows. I've always wanted to learn how to code a buffer overflow just couldn't find the resources.

Nice tut.

~Cheers

1 Reply

1

pry0cc Leader & Offsec Engineer & Forum Daddy Oct '17

Hi! Sweet article! Nice to see somebody using IDA 😛

Would you be able to fix the code formatting on the last script?

1

pry0cc Leader & Offsec Engineer & Forum Daddy Valentine Oct '17

Have you ever used the search bar? There are multiple articles on Stack Overflows / Buffer Overflows. Smashing the stack is the most covered exploitation topic.

4

Techno_Forg Zain Oct '17

Well than, skimmed it briefly and saw that it looked good. Only suggestion is maybe less pictures.

Looks good!

–Techno_Forg–

1

Sk0xic Oct '17

The idea is that what I say is as comprehensible as possible and that better than detailing everything with images, however thanks 😉

2

1 MONTH LATER

REal0day Nov '17

Great tutorial!
Hope they're more to come!

1

1 MONTH LATER

CLOSED DEC 22, '17

Reply

Suggested Topics

Topic	Replies	Activity
Windows 7 after the Supportend Exploit Development windows	13	4d

Topic	Replies	Activity
HackTheBox Writeup: Frolic Hackthebox Writeups	8	Mar 25
My first attempt at XSS Beginner Guides xss	2	May 24
HackTheBox Writeup: Arkham Hackthebox Writeups	3	29d
Latest Google Docs Dump that I can’t find anywhere anymore aug 15, 2019 Questions	5	12h

Want to read more? Browse other topics in Exploit Developm… **or view latest topics.**