Exploits and Their Effects

This chapter focuses on exploits and their detrimental effects on network security. Working of some severely damaging exploits is explained and countermeasures are suggested. Complete life cycle of vulnerability and exploit is explained.

An exploit is existence of a weakness; if a weakness is present, it can be exploited. More technical term used for the weakness is vulnerability. Vulnerability is a security flaw found within certain technology. The technology can be an operating system, all application program, a network protocol, a mathematical algorithm, or sometimes a hardware component, such as buffer memory of network interface card. An exploit is a computer program designed to take advantage of vulnerability. Objectives of an exploit range from as serious as getting the root access to simply access of files on the system.

1.1 Life Cycle of a Vulnerability and an Exploit

The term “life cycle” usually implies a series of stages through which an organism passes between recurrences of a primary stage, i.e. there exist a fixed and linear progression from one phase to the next. But vulnerability follows a different life cycle pattern and transits through distinct states: birth, discovery, and disclosure, release of fix, publication and automation of the exploitation.

Virtually every complex piece of code has some vulnerability in it. But users are unlikely to see attacks against their platforms until someone uncovers and discloses vulnerability. Initially product is iii stable state as released by the vendor; vulnerability is discovered by a user and reported to the vendor. It is rectified and product again goes into a stable mode as shown in Figure 4.1.
On the other hand, when vulnerability is discovered, instead of reporting it to vendor, it is reported in a public forum like the bug trap security mailing list. Although nearly all exploits start with public disclosure, the number attacks doesn’t really take off until, a talented hacker provides an automated tool that relatively unskilled vandals can use.

Intrusions increase once the community discovers vulnerability, with the rate of increase accelerating as news of the vulnerability reaches to wide spectrum of users. This trend continues until the vendor releases a patch or workaround. But mere release of patch does not decrease the intrusion rate until it is applied to the vulnerable software. The time between the vulnerability disclosed and the patch released to fix is very critical and this time needs to be minimized. The system oscillates between stable and vulnerable states. A Network Security framework should always try to pull system into to stable state as soon as possible.

**Figure 4.1: Vulnerability Life Cycle [172]**
There are many different categories of exploits that an attacker uses to attack a machine. Exploits are categorized depending upon the nature of their launch. Mainly exploits can be launched by using any of following techniques as shown in Figure 4.2.

![Exploit Categorization](image)

**Figure 4.2: Exploit Categorization [143]**

Minimal testing, fast time to market, poor coding principle etc., have lead software products to a walloping situation which could have been averted to maximum extent by applying sound Software Engineering Principles and Practices. There have been many causes to the vulnerabilities lurking in the software systems, but buffer overflows or buffer (stack) overruns have been accounted for most of these in a very critical manner.

### 1.2 Stack Overruns
In the past, lots of security breaches have occurred due to buffer overflow. Problems with buffer overflow have been identified as far back as 1960s. Major cause for these overflows/overruns is poor error checking, bound checking etc. Most of the new exploits are based on buffer overflow attacks. A buffer overflow attack is when an attacker tries to store too much information in an undersized receptacle. A common implementation is when a user of the program gives the program more data then the developers of the program allocated to store it.

In November 1988, many organizations had to cut themselves off from the internet because of the “Morris Worm”, which was a program written by 23-year-old Robest Tappan Morris to attack VAX and sun machines. By some estimates, this program took down 10% of the entire Internet. In July 2001, another worm named “Code Red” eventually exploited over 300,000 computers worldwide running Microsoft’s ITS Web Server.

In January 2003, the “Slammer” (also known as “Sapphire”) worm exploited vulnerability in Microsoft SQL Server 2000 software, disabling parts of the Internet in South Korea and Japan, disrupting Finnish phone service, and slowing many U.S. airline reservation systems, credit card networks, and automatic teller machines.

All of these attacks — and many others — exploited a vulnerability called a buffer overflow. A stack-based buffer overrun occurs when a buffer declared on the stack is overwritten by copying data larger than the buffer. Variables declared on the stack are located next to the return address for the function’s caller. The attacker can get a program with a buffer overrun to bind a command shell to a particular port so as to take command of the system.

A buffer is a contiguous allocated chunk of memory. In C and C++, there are no automatic bounds checking on the buffer, which means a user can write past a buffer. Buffers are usually implemented using arrays and memory allocation routines like malloc () and new. An extremely common kind of buffer is simply an array of characters. For example:

```c
int main ( ) {
    int buffer_space [10];
    buffer_space [20] = 10;
}
```
The above C program is a valid program, and every compiler can compile it without any error. However, the program attempts to write beyond the allocated memory for the buffer, which might result in unexpected behavior. Many computer processors, including all x86 processors, support stacks that grow “down” from higher to lower addresses. Every process starts running with the following three segments:

- **Text Segment** is a read-only part that includes all the program instructions. Assembly instructions, that are the equivalent of the High level language code, are included in the segment.

- **Data Segment** is the block where initialized and uninitialized (BSS) data is present. In the above code, `int buffer_space [10]` is uninitialized data so it is part of BSS.

- **Third segment** is called “Stack segment”, here dynamic variables are allocated and deallocated; and where return addresses for functions are stored temporarily.

![Figure 4.3: Memory layout of a Linux process](image)

A stack is a contiguous block of memory containing data. A stack pointer (SP) points to the top of the stack. Whenever a function call is made, the function parameters are pushed onto the stack from right to left. Then the return addresses (address to be executed after the function returns), followed by a frame pointer (FP), and is pushed on the stack. A frame pointer is used to reference the local variables and the function parameters, because they are at a constant distance from the FP. Local automatic variables are pushed after the FP. Thus, every time a
function calls another function, more data will be added to the left (lower addresses) until the system runs out of memory for the stack.

Static variables are loaded to the data segments part of the program, whereas dynamic variables are allocated and deallocated within the stack region of the executable in the memory. And, “stack-based” buffer overflows occur here, when user stuff more data than a data structure, say an array can hold, data exceed the boundaries of the array overriding important data.

```c
void func(int a, int b, int c)
{
    char buff [5];
    gets(buff);
}

int main()
{
    func(1,2,3);
    return 0;
}
```
When main() called func(), it pushed the values for c, then ‘b. then ‘a’ onto the stack. It then pushed the return (ret) value, which tells ftmnc() where to return to in main() once func() is completed. It also recorded the “saved frame pointer” (sfp) onto the stack; Once func() started up, it sets aside space for buff(), which has a lower address location.

Now if more data is input than buff() can handle. As their is no automatic boundary checking, the next values will go to the “next” locations in memory. That means that the attacker can overwrite sfp (the saved frame pointer) and then overwrite ret (the return address). Then, when func() is completed, it will “return’, but instead of returning to main (), it will return to whatever code the attacker wants to run, thus causing this vulnerability to be exploited.

Often the attacker will overrun the buffer with the malicious code the attacker wants to run, and the attacker will then change the return value to point to the malicious code they have sent. There are other ways to exploit buffer overflows besides smashing the stack and changing the return address. Instead of overwriting the return address, attacker could smash the stack (overflow a buffer on the stack) and then overwrite local variables to create an exploit.

The buffer need not the stack at all, dynamically allocated memory in the heal of also called the (“malloc” or ‘new” area), or in some statically located memory such as ‘Global” and “static memory. Basically, if an attacker can overflow the bounds of a buffer, code is vulnerable. However, the most dangerous buffer overflow attacks are stack-smashing attacks, because it’s especially easy for an attacker to gain control over an entire machine if a program is vulnerable.

1.2.1 Exploit Development Process

In this work, metasploit framework huts been used to study the exploits, know their working and analyze them in detail. This kind of framework gives the security administrator, the opportunity to test systems for security weaknesses before an attacker discovers this. In fact these types of utilities may eventually become common practice for system developers to rise
while writing the application and this may stop the vulnerability from ever being published in
time first place.

The basic format for exploiting the system is as follows:

- Pick which exploit to rise
- Configure the exploit with remote IP address and remote port number
- Pick a payload
- Configure the payload with local IP address and local port number
- Execute the exploit

The console of MSF (Metasploit Framework) work similar to Command Line Interface (CLI) of an CISCO device. The msfconsole interactive command—line interface provides a command set that allows to manipulate the framework environment, set exploit options, and ultimately deploy the exploit. Windows operating system has been most widely deployed and many windows web-server deployments have faced the wrath of Internet worms. These worms exploited some vulnerability in the Internet Information Server (IIS) and its related code implementations. In the experimentation given below, research effect were concentrated on windows 2000 with the IIS version 5.0

1.2.2 Life Cycle of a Typical Windows Exploit

Windows 2000 introduced native support for the Internet Printing Protocol (IPP), an industry-standard protocol for submitting and controlling print jobs over HTTP. The protocol is implemented in Windows 2000 via an ISAPI extension that is installed by default as part of Windows 2000 but which can only be accessed via IIS 5.0.

Security vulnerability results because the ISAPI extension contains an unchecked buffer in a section of code that handles input parameters. This could enable a remote attacker to conduct a buffer overrun attack and cause code of choice to run on the server. Such code would run in the Local System Security context. This would give the attacker complete control of the server. The attacker could exploit the vulnerability against an server with which a web session can be conducted. No other services would need to be available, and only port 88 (HTTP) or 413 (HTTPS) would need to be open. Following public forums and lists announced thus vulnerability:
This is a very serious vulnerability and following sections analyze its working and effects on network security.

1.2.3 Finding the Attack Vector

An attack vector is the means by which an attacker gains access to a system to deliver a specially crafted payload. This payload can contain arbitrary code that gets executed on the targeted system. The first step in writing an exploit is to determine the specific attack vector against the target host. Because Microsoft’s ITSS Web server is a closed-source application, research work relied on security advisories and attempted to gather as much information as possible. The better understand the buffer overflow exploits as show in the Figure 4.5 was used.
The vulnerability to be triggered in the exploit is iis50_printer_overflow. It exploits a buffer overflow in the request processor of the Internet Printing Protocol ISAPI module in ITS. This module works against Windows 2000 service pack 0 and 1. Second step is to calculate the offset of the overflow vulnerability. After deciding on the most reliable control vector, a valid return address must be found. Character and size limitations will need to be resolved before selecting a payload. A nop sled must be created. Finally, the payload must be selected, generated, and encoded.

When ITS receives a request, it passes the filename to the ISM dynamically linked library (DLL) for processing. Because neither the IIS server nor the ISM DLL performs bounds checking, it is possible to overflow a buffer in a vulnerable function and overwrite the return address. By hijacking the flow of execution in the ISM DLL and subsequently the inetinfo.exe process, the attacker can direct the system to execute the payload.

A standard request for a Web page consists of a GET or POST directive, the path and filename of the page being requested, and HTTP protocol information. The request is terminated with two newline and carriage return combinations (ASCII characters 0x10 and 0x13, respectively). The following code snippet shows a GET request for the index.html page using the HTTP 1.0 protocol.

```
GET /index.html HTTP/1.0
```

Port scanning with the help of nmap tool was cloned in order to find whether the victim machine is vulnerable or not. Port scanning showed the port 88 and 443 are open. Banner grabbing was used to check the system’s vulnerability. IIS version 6 installed on freshly installed Windows 2000 machine as shown in Figure 4.6 shows system is vulnerable, whereas system with ITS version 6.1 as shown in Figure 4.7 shows system is not vulnerable. During the process of banner grabbing, network traffic was also recorded with the help of Wireshark (a Network Traffic Sniffing software).
Bairner Grabbing code snippet used is:

telnet 172.31.1.90 80 GET /NULL. Printer

The overflow happens in the isapi handling the printer extension. The vulnerability arises when a buffer of approximately 420 bytes is sent with the HTTP Host: header for a printer ISAPI request.

At this point all attacker has successfully caused buffer overflow within ITS and has overwritten EIP. Now normally the web server would stop responding once “buffer overflowed it. However, windows 2000 will automatically restart the web server notices that the web server has crashed. While the feature is nice to help create a longer period of “up time” it is actually a feature that makes it easier for remote attacks to execute code against Windows 2000 ITS 5.0 web servers.
Knowing the attack vector, a. Pen script to overflow the buffer and overwrite the return address was written. Payload was generated using MSF web interface.

$string= "GET /NULL. printer HTTP/1.0\nHost:" ;

$string.= "\x90" x 268;

$string .= "MMMM";

$string .= "B" x 241;

$string = "\n\n"

open (NC, "Inc.exe 172.31.1.90 80");

print NC $string;

close (NC)
Figure 4.8: First Attack String after appropriate buffer calculations [147]

Figure 4.8 shows the first attack string, buffer calculations were done by making script to autorun with unique attack strings. To get the code executed on the victim machine, activeperl was used on the attacker machine and output was sent to the victim machine using netcat. A pipe is created between the NC file handle and the Netcat utility. The Netcat utility has been instructed to connect to the target host at 172.31.1.90 on port 88. The $string data is printed to the NC file handle. The NC file handle then passes the string data through the pipe to Netcat which then forwards the request to the target host. After sending the attack string, to verify that the return address was overwritten a debugger OllyDbg was attached to the ITS process, inetinfo.exe. OllyDbg screen showing register status of inetinfo.exe after buffer overflow ix shown in Figure 4.9.

Figure 4.9: IIS 5.0 attached to OllyDbg for debugging [149]

As can be seen from the OllyDbg output, it is possible to overwrite the saved return address with an arbitrary value. Because the return address gets popped into EIP, an exploit can control the EIP register. Controlling EIP will allow an exploit to lead the process to the payload, and therefore, it will be possible to execute any code on the remote system. The term payload
refers to the architecture-specific assembly code is passed to the target in the attack string and executed by the target host. A payload is created to cause the process to produce an intended result such as executing a command or attaching a shell to a listening port.

1.2.4 Selecting a Control Vector

The control vector is the path through which the flow of execution is directed to exploit code. In a buffer overflow attack that overwrites the return address, there are generally two ways to pass control to the payload. The first method overwrites the saved turn address with the address of the payload on the stack: the second method overwrite the saved return address with an address inside a shared library. The instruction pointed to by the address in the shared library causes the process to bounce into the payload on the stack.

The first technique overwrites the saved return address with an address of payload located on the stack. As the processor leaves the vulnerable function, the return ass is popped into the EIP register, which now contains the address of the payload. EIP point to where the flow of execution is going next. By getting the address of the payload into EIP, program is redirected to follow the flow of execution of an exploit.

This technique is suited best for the Unix systems. Shared libraries in UNIX do riot specify preferred base addresses, so in UNIX the shared library trampoline method is not as reliable as the direct stack return. The base address of the Windows stack is not as predictable as the base address of the stack found on UNIX systems. In case of Windows operating system, use of shared library trampoline has become useful workaround. The process involved with the shared library method is somewhat more complex than returning directly to the stack. Instead of overwriting the return address with an address on the stack, the return address is overwritten with the address of an instruction that will copy the value of the register pointing to the payload into the EIP register.

By design, there exist many instructions that modify EIP, including CALL, JMP, and others. The CALL instruction is used to alter the path of execution by changing the value of EIP with the argument passed to it. The CALL instruction can take two types of arguments: a memory address or a register. If a memory address is passed, then CALL will set the EIP register equal to that address. If a register is passed, then CALL will set the EIP register to be equal to the value within the argument register. With both types of arguments, the execution path can be
controlled. As it is not possible to consistently predict stack memory addresses in Windows, so a register argument must be used.

### 1.2.5 Finding and Using a Return Address

When developing an exploit, there may be variations between versions, so it is important to find the right offsets for each. Finding a return address involves, selection of instruction to be executed, the opcodes for the instruction, which DLLs are loaded by the target application and search for the specific opcodes through the memory regions mapped to the DLLs that are loaded by the application. The Metasploit Opcode Database contains over 7.5 million precalculated memory addresses for nearly 250 opcode types and can be used to find the addresses. The exploit can now be updated to overwrite the saved return address with the address of the CALL instruction. The target host runs on anx86 architecture, the address must be represented in little endian format. It can be achieved by using pack instruction in pen as shown below:

```
$retaddr = “0x732c45f3”;
$string = “\x90” x 280; substr($string, 268, 4 “$retaddr”);
substr($string, 268, 4, pack( “V”, $retaddr));

# payload is at: [ebx + 96] + 256 + 64

$string .= “\x8b\x4b\x60”; // mov ecx, [ebx + 96]

$string “\x80\xc5\x01”; // add ch, l.

$string “\xff\xel”; // imp ecx
```

### 1.2.6 Finding Bad Characters

Many applications perform filtering on the input that they receive, so before sending a payload to a target, it is important to determine if there are any characters that will be removed. A test string can be created that contains all possible ASCII characters represented by
values from 0 to 255 or this test string can be repeated in the free space around the attack strings return address while the return address is overwritten with an invalid memory address. After the return address is popped into EIP, the process will halt on air access violation: now the debugger can be used to examine the attack string in memory to see which characters were filtered and which characters caused early termination of the string. One value that virtually always truncates a string is 0x00 (the NULL character). A bad character test string usually does not include this byte at all. If a character prematurely terminates the test string, then it must be removed and the bad character string must be sent over again until all the bad characters are found.

**Nop Sleds**

EIP must land exactly on the first instruction of a payload in order for it to execute correctly. Because it is difficult to address of the payload stack address of between systems, prefix the payload with a 110 operation (nop) sled. A nop sled is a series of 1101) instructions that allow EIP to slide down to the payload regardless of where EIP lands on the sled. By using a nop sled, an exploit increases the probability of successful exploitation because it extends the area where EIP can land while also maintaining the process state. Preserving process state is important because the same preconditions to be true before payload executes no matter where EIP lands. Process state preservation can be accomplished by the 110 instruction because the 110 instruction tells the process to perform no operation. The processor simply wastes a cycle and moves the next instruction, and other that incrementing EIP, this instruction does not modify the state of the process. Every CPU has one or more opcodes that can be used as no-op instructions. The x86 CPU has the 110 opcode, which maps to 0x90. To extend the landing area on an x86 target, a payload could be prepended with a series of 0x90 bytes. 0x90 represents the XCHG EAX. EAX instruction which exchanges the value of the EAX register with the value in the EAX register, thus maintaining the state of the process.

### 1.2.7 Choosing a Payload and Encoder

The final stage of the exploit development process involves the creation and encoding of a payload that will be inserted into the attack string and sent to the target to be executed. A payload consists of a succession of assembly instructions which achieve a specific result 011 target lost such as executing a command or opening a listening connection that returns a shell.
The latest release of the Metasploit Framework includes over 65 payloads that cover mine operating systems 011 four architectures. Research work used Metasploit framework’s payload database to generate and use win32_bind shellcode. Figure 4.10, shows the execution of iis50_printer_overflow exploit from within MetaSploit Framework.

![MsfConsole](image)

Figure 4.10: Successful Exploitation of Winclows2000 SP 4 [124]

1.2.8 Suggested Overflow Defense Methods

1.2.8.1 Canary-Based Defenses

Researcher Crispen Cowan created an approach called StackGuard. Stackguard modifies the C compiler (gcc) so that a “canary” value is inserted in front of return addresses. The Lca1ary acts like a canary in a coal mine: it warns when something has gone wrong. Before any function returns, it checks to make sure that the canary value hasn’t changed. If an attacker overwrites the return address (as part of a stack-smashing attack), the canary’s value will probably change and the system can stop instead. This is a useful approach, but this does not protect against buffer overflows overwriting other values (which may still be vulnerable). There has been research effort to extend this approach to protecting other values (such as those on the heap) as well. Stackguard is used by Immunix (a Secure Linux Clone).

IBM’s stack-smashing protector (ssp), originally named ProPolice. is a variation of StackGuard’s approach. Like StackGuard, ssp uses a modified compiler (gcc) to insert a canary in function calls to detect stack overflows. It reorders where local variables are stored, and copies
pointers in function arguments, so that they’re also before any arrays. This strengthens the protection of ssp: this means a buffer overflow can’t modify a pointer value (otherwise an attacker who can control a pointer can control where the program saves data using the pointer). Microsoft has added a compiler flag (/GS) to implement canaries in its C compiler, based on the StackGuard work.

1.2.8.2 Non-Executing Stack Defense

Another approach starts by making it impossible to execute code on the stack. Unfortunately, the memory protection mechanisms of the x86 processors (the most common processors) do not easily support this: normally if a page is readable, it is executable. It turns out that there are cases where executable programs are needed on the stack; this includes signal handling and trampoline handling. Trampolines are exotic constructs sometimes generated by compilers to support constructs like nested subroutines. The original patch to have Non-executing stack defenses in Linux was rejected by Linus Torvalds in 1998. Even if code can not be placed on the stack, an attacker could use a buffer overflow to make a program “return” to an existing subroutine (such as a routine in the C library) and create an attack i.e. just having a non-executable stack isn’t enough strength to protect against buffer overflow attacks.

Another idea was developed to counter this problem by moving all executable code to all area of memory called the “ASCII armor” region. Attackers often can not insert the ASCII NUL character (0) using typical buffer overflow attacks. That means that attackers find it difficult to make a program return to an address with a zero in it. Since that is the case, moving all executable code to addresses with a 0 in it makes attacking the program far more difficult. The largest contiguous memory range with this property is the set of memory addresses from 0 through 0x01010100, so that has christened the ASCII armor region. Combined with non-executable stacks, this approach is valuable: non-executable stacks prevent attackers from sending new executable code, and ASCII-armor makes it hard for attackers to work around it by exploiting existing code. This protects against, stack, buffer and function pointer overflows, all without recompilation. However, ASCII-armor does not work for all programs: big programs may not, fit in the ASCII-armor region (so the protection will be imperfect), and sometimes attackers can get a 0 into their destination. Also, sonic implementation does not support trampolines, so the protection may have to be disabled for programs that need them. Red Hat’s Ingo Molnar implemented this idea in “exec-shield” patch which is used by Fedora core (the
freely available distribution available from Red Hat). The latest version OpenWall GNU/Linux (OWL) uses an implementation of this approach by Solar Designer.

1.2.8.3 Other approaches

One approach is to make standard library routines more resistant to attack. Lucent Technologies developed Libsafe, a wrapper of several standard C library functions like strcpy() known to be vulnerable to stack-smashing attacks. Libsafe is open source software licensed under the LGPL. The libsafe versions of those functions check to make sure that array overwrites can’t exceed the stack frame. However, this approach only protects those specific functions, not stack overflow vulnerabilities in general, and it only protects the stack, not local values in the stack. Their original implementation uses LD_PRELOAD, which can conflict with other programs.

Another approach is called “split, control and data stack”, the idea is to split the stack into two stacks, one to store control information (such as the “return” address) and the other for all the other data. Xu et al. implement this in gee, and StackShield implements it in the assembler. This makes it much harder to manipulate the return address, but it does not defend against buffer overflow attacks that change the data of calling functions.

1.3 Denial of Service (DoS) Attacks

Another powerful attack which has many exploits publicly available is Denial of Service. A Denial of Service attack is attack through which a person can render a system unusable or significantly slow down the system for legitimate users by overloading the resources so no one else can access it. It can be an accidental or deliberate in nature. These attacks are very difficult to prevent. However, restricting access to critical accounts, resources and files can be helpful in curtailing these.

There are two types of DoS attacks. One type of attack crashes a system and/or network. Another type of attack involves flooding the system and/or network with so much of information that it can not respond to the legitimate requests. With these traditional Denial of Service attacks, a single machine is usually launching the attack against the victim and it is very much feasible to block attackers IP at the Firewall level. On the other hand, when hundreds of machines under the control of some master launch the attack against a victim it is called Distributed Denial of Service (DDoS) attack and it is very difficult to protect against DDoS. DDoS scenario is depicted in Figure 4.11.
1.3.1 Ping of Death

PING is used interactively on the (command line to test. readability of a remote computer. The PING based Distributed Denial of Service (DDoS) attacks arc known to be quite damaging to the availability of the web-based services. The PING-attack floods the victim (a computer/server) with ICMP echo request messages, in an attempt to tie up its link bandwidth and computing resources. The PING request message uses ICMP echo-request message, whereas the PING reply message uses ICMP echo reply message. The ICMP standard is defined in RFC-792 [90] and updated in RFC-1122 [136]. In October of 2002, a massive Distributed Denial of Service (DDoS) attack interrupted web traffic on nine of the 13 DNS root servers that control the Internet. The 13 DNS root servers are the backbone that, runs the domain names and IP addresses on the web. The DDoS attack lasted for 1 hr. and was clone via PING-flooding of the root servers.

The Internet Software Consortium (ISC), which manager one of the targeted root servers, reported 80 Mbps of traffic to its server, which was more than 10 times the normal load. During the course of the PING attack, only four of 13 roots, servers remained up arid were running while seven were completely crippled. Coming on the heels of cyber terrorism threats,
this PING-based DDoS attack has been taken seriously as it was an attack on the Internet itself. Earlier, similar PING-flooding based DDoS attacks have brought down high profile web sites such as Ebey, Etrade and Yahoo [64].

Such exploits are real dangers, one such attack “Ping of Death” works by sending a very large ping packet to the host machine. This exploit works at network level and since many operating environment does not deal with such an exceptional condition, it causes most of machines especially running windows 98 to hang or crash. This also causes blue screen of death to appear in case of Windows NT. It uses large Internet Control Message Protocol (ICMP) and works at layer 3 in the OSI model. ICMP is usually used to convey status and error information which includes network transport and network congestion problems.

This attack is very simple to initiate by ping program as shown below:

C:\>ping -1 1500 172.31.1.91 -t

Pinging 172.31.1.91 with 1500 bytes of data:

Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128

Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128

Reply from 172.31.1.91: bytes=1500 time=2ms TTL=123

Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128

Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128

Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128

Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128

Ping statistics for 172.31.1.91:

Packets: Sent = 7, Received = 7, Lost = 0 (0% loss),

Approximate round trip times in mini-seconds:

Minimum 2ms, Maximum = 2ms, Average = 2ms
By default Windows ping program sends payload of just 32 bytes, but by using -1 option attacker can specify the load up to 65500 bytes.

**Signature of the Attack**

The following is the Wireshark (a Sniffing tool) output showing the network traffic clump of ping -1 65500 172.31.1.91 -t

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Fragmented IP protocol (proto=ICMP 0x0, off =0) [Reassembled in #45]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
<td>172.31.1.4</td>
<td>172.31.1.91</td>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.001051</td>
<td>172.31.1.4</td>
<td>172.31.1.91</td>
<td>IP</td>
<td>Fragmented IP protocol (proto=ICMP 0x0, off =1480) [Reassembled in #45]</td>
</tr>
<tr>
<td>3</td>
<td>0.002099</td>
<td>172.31.1.4</td>
<td>172.31.1.91</td>
<td>IP</td>
<td>Fragmented IP protocol (proto=ICMP 0x0, off =2960) [Reassembled in #45]</td>
</tr>
<tr>
<td>44</td>
<td>0.045220</td>
<td>172.31.1.4</td>
<td>172.31.1.91</td>
<td>IP</td>
<td>Fragmented IP protocol (proto=ICMP 0x0, off =63640) [Reassembled in #45]</td>
</tr>
</tbody>
</table>
Frame 45 (422 bytes on wire, 422 bytes captured) Ethernet II, Src:
Dst: 172.31.1.91 (172.31.1.91)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
Total Length: 408
Identification: 0x05e2 (1506)
Flags: 0x00
Fragment offset: 65120
Time to live: 128
Protocol: ICMP (0x01)
Header checksum: 0xb919 [correct]
Source: 172.31.1.4(172.31.1.4)
Destination: 172.31.1.91 (172.31.1.91)

IP Fragments (65508 bytes): #1(1480), #2(1480), #3(1480), #4(1480), #5(1480),
...........#44(1480), #45(388)

Internet Control Message Protocol

Type: 8 (Echo (ping) request)
As payload can not be greater than 1480 bytes it is fragmented into number of requests and gets assembled at the destination thus causing the victim to hang or crash. Above details show 65500 payloads is divided into 45 IP fragments numbered from #1 to #45 with final fragment having payload of 388 bytes.

There are two ways to get defended against Ping of Death, one is to patch the system with latest version or block the large ping payloads at the routers or firewalls i.e. network peripheries.

### 1.3.2 Land Exploit

Land exploit launches a Denial of Service attack against various TCP implementations. The program sends a TCP SYN packet where the source aid destination addresses are same as well as source and destination ports are same. This represents an exceptional condition thus landing system into a crash situation.

TCP is a reliable connection—oriented protocol that operates at transport layer i.e. layer 4 of OSI model. It requires a three- way handshake to initiate new connections. When a new connection is initiated, a. SYN flag is made on to make machines synchronize. In reply server machine sends a packet with SYN and ACK flags on. But in case of Land Exploit as source and destination addresses are same; most machines will crash or hang because their TCP implementations are not aware of such situations.

**Signature of the Attack**

Any packet that has following characteristics is a land attack:

- Source and Destination addresses are same
- Source and Destination ports are same
These generally do not occur, so any packets that have these features should be dropped.

1.3.3 Smurf

The smurf’ attack is quite simple. It’ has a list of broadcast addresses winch it stores into au array, and sends a spoofed ICMP echo request to each of those addresses in series and starts again. It falls under category of network-level attacks against hosts. A perpetrator sends a large amount of ICMP echo (ping) traffic at broadcast addresses, all of it having a spoofed source address of a victim. If the routing device delivering traffic to those broadcast addresses performs the IP broadcast to layer 2 broadcast, most hosts on that IP network will take the ICI\IP echo request and reply to it with an echo reply each, multiplying the traffic by the number of hosts responding. On a multi—access broadcast network, there could potentially be hundreds of machines to reply to each packet. The two main components of the smurf attack are the use of forged packets and the use of broadcast address. A broadcast address is a single address used to send a packet to all hosts on a network segment. The IP broadcast address for test bed network 172.31.1.0/24 is 172.31.1.255. This address will send the packet to all the machines on 172.31.1.0 network.

To start this attack, the attacker generates an ICMP echo request using the forged source address and a broadcast address as the destination. This causes the packet to be sent to all machines on that network segment, with each machine replying to the request and sending an ICMP echo reply back. This leads to high network traffic thus causing degraded performance or denial of service for that network segment.

Signature of the Attack

One signature of the attack is degraded network performance. Most Intrusion Detection system (IDS) look for large number of’ ICMP requests organization from specific host and an ICMP request sent to broadcast address. Figure 4.12 shows tile network traffic capture where attacker machine 172.31.1.17 forged the IP address to 172.31.1.90, which is the victim of broadcast ICMP traffic.
The amount of bandwidth and packets per second (pps) that can be generated by this attack is quite large. To stop routers from converting layer 3 broadcasts into layer 2 broadcasts, use the “no ip directed—broadcast interface configuration command. This should be configured on all routers which provide routing to large multi—access broadcast networks (generally LANs).

Filtering ICMP echo reply packets destined for your high—profile machines at the ingress interfaces of the network border routers will then permit the packets to be dropped at the earliest possible point. However, it does not mean that the network access pipes would not fill, as the packets will still come down the pipe to be dropped at the router. It will, however, take the load off the system being attacked.

1.3.4 SYN Flood

SYN Flood exploits vulnerability in TCP implementation (s) by opening a large number of half—opens TCP/IP connections. The impact of this can be observed on most, of the operating systems. TCP/IP three—way handshake happens as shown below:

- Client initiates a connection with the server. Client sends a packet to the server with SYN bit set.
- The server receives this packet and sends back a packet with the SYN bit and an ISN (Initial Sequence Number) for the server.
• Client sets the ACIK bit acknowledging the receipt of the packet and increments the sequence number by 1.

Following the above sequence of steps, the two machines have successfully established a session. Sequence Numbers are very important to provide reliable communication but they are also crucial to hijacking a session. Sequence numbers are a 32-bit counter, which means the value can be any of over 4 billion possible compilations. The sequence numbers are used to tell the receiving machine what order the packets should go in when they are received. A typical TCP/IP connection happens when a system connects to some site using HTTP, Telnet or SSH. Shown below is network traffic clump showing three-way handshake where a client 172.31.1.4 establishes a http connection to the web server 172.31.1.90 at port 88.

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6.055714</td>
<td>172.31.1.4</td>
<td>172.31.1.90</td>
<td>TCP</td>
<td>1294 &gt; http [SYN] Seq=0 Len=0 MSS=1460</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6.084340</td>
<td>172.31.1.90</td>
<td>172.31.1.4</td>
<td>TCP</td>
<td>http &gt; 1294 [SYN, ACKI Seq=0 Ack=1 Win=17520 Len=0 MSS=146</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6.084445</td>
<td>172.31.1.4</td>
<td>172.31.1.90</td>
<td>TCP</td>
<td>1294 &gt; http [ACK] Seq=1 Ack=1 Win=65535 Len=0</td>
</tr>
</tbody>
</table>

First phase of the three-way handshake SYN bit is set on b 0x0002, source is 172.31.1.4 with src port (1294) and destination is 172.31.1.90 with dst port (80).

Frame 4 (62 bytes on wire, 62 bytes captured) Internet Protocol,

Src: 172.31.1.4 (172.31.1.4), Dst: 172.31.1.90 (172.31.1.90)

Transmission Control Protocol, Src Port: 1294 (1294), Dst Port: http (80), Seq: 0, Len: 0

Source port: 1294 (1294)
Destination port: http (80)

Sequence number: 0 (relative sequence number)

Header length: 28 bytes

Flags: 0x0002 (SYN)

......1. =Syn: Set

Window size: 65535

Checksum: 0x5dd3 [correct]

Options: (8 bytes)

**Signature of the Attack**

The signature of the attack is fairly simple. When a large number of SYN packets appear on a network without the corresponding reply packets as shown in Flow Graph Figure 4.13 system is under SYN flood attack.

![Flow Graph](image)

*Figure 4.13: SYN Flood Attack Signature [176]*
This attack causes a number of SYN packets sent to the destination. As SYN_ACK is received from the server attacker machine does not sent the ACK packet back thus three-way handshake is not complete, these are called half-open connections. The server has in its system memory a built-in data structure describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many half-opened connections.

The memory area pertaining to this data structure eventually gets filled and the system is unable to accept any new connection. Normally, TCP implementations use timeout to expire such connection, but in case(s) of flooding attacker sends SYN packets faster than that can be replied by the victim. Thus impacting the performance severely, and in some cases might, even crash the system.

Proper router and/or firewall configurations can reduce the likelihood of such attacks by allowing only a limited number of half-opened connections to be active at a given time. But in case of spoofed IPs by the attacker, unknowingly legitimate user may be blocked to access the service(s). Linux and Solaris operating environments implement SYN cookies to curtail such floods. SYN cookies, stop storing the information in the queue by setting the ISN (initial sequence number) as a function of the sender’s IP address.

1.4 A Live Walk-Through into System Hacking

This work presents a live walk-through where an attacker goes through a series of steps towards a vulnerable (Non-patched) system. The steps involved are base-lined against DCOM-RPC (Distributed Component Object Model - Remote Procedure Call) exploit, which is very common and hits most of the Windows OS environments.

This is typical case of an educational institute where say Mr. Cracker comes with his laptop, hooks on the laptop to the free Info-outlet port and gets an IP (Internet protocol) address dynamically assigned by Institute’s DHCP (Dynamic Host Configuration Protocol). Following series of steps are followed to gain access to critical systems.

1.4.1 Reconnaissance and Enumeration Phase

This is the first phase of an attack hacker tries to gather as much as information possible for the target. There are two ways of gathering information first one is passive where au
attacker listens to the network traffic by using a Network Sniffing Software and secondly information can be gathered by probing the machine/network thus leading to an active methodology. Whatever may be the case, intent is to know which operating system in running on the target and which all ports are open so as to tailor made an attack.

One of the most popular types of passive attacks is sniffing. This involves sitting oil a network segment, watching and recording all traffic that passes on the segment. This will provide lot of information to the hacker. Hacker can sniff NT authentication packets and later on use some password cracking tools to get user credentials. In active reconnaissance attacker probes the system with some tool.

Attacker uses a tool to scan the target and enumerate so as to expose manly critical details which helps to mould the attack accordingly. SuperScan is used to scan the whole network so as to build an inventory of the systems running on the network and finally targeting the weakest among these to launch the attack. For this live walk-through a machine with 192.168.1.75 (private IP series address) is chosen to be an attacker’s machine and 192.168.1.76 as IP address for the victim. Attacker launches scanning (i.e. active reconnaissance) as in shown in Figure 4.14.

![Figure 4.14: Active Reconnaissance: Probing the system with a tool](image-url)
From this hacker comes to know that the victim machine is having ports 135, 137 opened, which are basically used by windows NetBIOS over TCP/IP for file sharing. Next hacker runs enumeration for this particular machine so as to get more details about the accounts, shares, services etc. The following information is retrieved by enumeration as shown in Figure 4.15. This information is very critical and gives valuable information to the hacker.

![Figure 4.15: Enumeration Phase of the Victim Machine][165]

Attempting a NULL session connection on 192.168.1.76

NULL session successful to \192.168.1.76\IPC$

A null session is only established when there are no credentials for a process to start under (no user name or password). Typically, only the operating system itself runs as a system user.

Workstation/server type on 192.168.1.76

Windows 2000

Workstation/Server Name : “192.168.1.76”

Platform ID : 500

Version : 5.0
It also tells the hacker that the Operating System is Windows 2000 so that attacks can be tailored. Another important information shown is about the users, their names, password aging policy, last logon, number of logons etc.

Total Users: 2

---1---

Admin “Administrator”

Full Name: “ ”

System Comment: “Built—in account for administering the computer/domain”

User Comment: “ ”

Last logon: Sun Jan 08 14:44:12 2006 (0 days ago)

Password expires: Never

Password changed: 0 days ago

Locked out: No

Disabled: No

Number of logons: 1

Bad password count: 0

---2---

User “Guest”

Full Name: “ ”

System Comment: “Built—in account for guest access to the computer/domain”
User Comment: “ ”

Last logon: Never

Password expires: Never

Password changed: Never

Locked out: No

Disabled: Yes

Number of logons: 0

Bad password count: 0

Password and account policies on 192.168.1.76

Account lockout threshold is 0
Minimum password length is 0
Maximum password age is 42 days

As can be seen Account lockout threshold is by default set to 0, which means intruder can try out credentials any number times and will never be locked out.

Shares on 192.168.1.76

IPC: IPC$ (Remote IPC)

Disk: ADMIN$ (Remote Admin)

Disk: C$ (Default share)

This shows default shares on the victim's machine. This much information is good enough for the hacker to launch attack on the system, install some Trojan so that lie can create hack door on the machine and later on can attach to the machine with greater ease.

1.4.2 Attack Phase

In the attack phase hacker uses tools to exploit the R.PC vulnerability and then netcat to get the victim machines prompt as shown in Figure 4.16.
dcomexploit 1 192.168.1.76

Now hacker uses netcat to connect to 192.168.1.76 at port number 4444. Netcat is very popularly known as swiff army knife tool for its versatility to make net connection across hosts.

netcat 192.168.1.76 4444

Next step is to gather data from the SAM database and pass it on the hackers machine. This is easily done with the help of pwdnmp3 tool, which dumps database as an output file, which hacker later on analyzes locally using dictionary based and/or brute force attacks.

1.4.3 Analysis of Hacked Data

Analysis of the captured data from the victimized machine can lead to cracking of passwords and hacker makes repository and goes to other machine for executing the same step of attacks. Attacker uses “Cain & able” tool to crack the LM and NT hashes as shown in Figure 4.17. Tool takes few minutes only to crack weak passwords. These weak passwords can become serious security loopholes and can be used later to crack the system.
Now hacker can deploy a trojan on this host so that later on he can log on to the machine using a backdoor. One can say once hacker got the password why does not he destroy the system. This all depends upon the criticality of a system. Actually this is not the aim of hackers for a low profiled system. Hackers want to create Botnets for themselves so that later on they can utilize these kinds of zombies to launch attacks on more critical networks. So today if a home user says “I do not have any confidential data on machine why should somebody bother to hack me”. This is total misconception, hackers use machines as launching pad for more serious attacks or to utilize computational power on these zombies to crack passwords using brute force methods.

1.5 Conclusions

Analysis of various critical exploits and their detrimental effects on network security is performed and outcomes are reported in this chapter. A complete exploit development process is demonstrated, which highlighted various phases through which an exploit is executed, namely, finding the attack vector, selecting the control vector, finding and using the return address. Finding bad characters, choosing the payload and choosing the encoder. Stack overrun defense methods and Denial of Service countermeasure are suggested. Attack signatures of Ping of Death, Land Exploit. Smurf and SYN flood are captured and reported. Finally, a live walk-through into system hacking is demonstrated, which highlighted the patching defense mechanism and emphasized imparting of security awareness and education to the users of the network.

This chapter achieves first objective of the work and gives insight into life cycle of an exploit and vulnerability, their development process and focuses on most critical exploits. Next
chapter, Analysis of Open source Counteract, focus(es) on the proactive methodologies like Counteract to uncover the tools and tactics of blackhat community, economy driving the botnets, and highlights synergy between reactive and proactive methods to bring security in the network fabric not just at the periphery.