

An Overview of Methods Employed by Hackers and Crackers

Pradeep Padala

Department of Computer Science & Engineering
Motilal Nehru National Institute of Technology
Allahabad, India 211004

Abstract—This paper describes how hackers and crackers break into systems and manage to compromise their security. It discusses some of the security problems that have surfaced as a result of poor design in the recent past, and that are likely to show up in the future. Bad design of operating systems, inherent flaws in the design of protocols and vulnerabilities in utility programs provide gateways to hackers and crackers. We describe these methods in detail and propose some defenses against these methods.

I. INTRODUCTION

Everyday, round the globe, umpteen computer networks and hosts are being broken into and are being compromised. The level of sophistication of these attacks varies widely. While it is generally believed that most break-ins succeed due to weak passwords, dummy login screens, social engineering etc., there still are a large number of intrusions that use more advanced techniques. We mainly deal with the flaws that occur due to bad design and implementation of systems and we have made an attempt to propose a new approach to system security.

In this paper we discuss how the design and implementation of reliable system can be enhanced. In the following, first we describe conventional methods. Then, some commonly seen flaws are examined. Finally, some techniques to guard against these attacks are explained.

II. CONVENTIONAL METHODS

The methods used by conventional hackers and crackers are simple and easy to employ. Most of the tools are automated and require little monitoring by the user. The hackers first make a database of IP addresses, allow their scanners to invade the Internet for these specific addresses and get the results. These tools are available on the Internet on numerous ftp sites. These tools are very much diverse in nature and some of these check for multiple vulnerabilities too.

The more sophisticated hackers implement Trojan horses and back-doors once they compromise a system. Backdoors allow easy access to the system whenever the user wants. The Trojans make the intruder undetectable. The Trojans make the intruder undetectable. He would not show up in the logs, system processes or file structure.

All these weaknesses are a direct consequence of flaws inherent in the design and implementation of the systems.

III. FLAWS IN THE DESIGN OF OPERATING SYSTEMS

We first deal with the core of the operating system, which interacts with hardware and sits in between the user and the hardware - the kernel. Hackers manage to break into systems due to the inherent problems in Network Operating Systems(NOS)[1]. A few of these methods were rampant in the past and most systems have been made impervious to such attacks. The others are very recent and should not be treated with levity. Some of the vulnerabilities are discussed below.

A. Inode Count integer overflow

Member `i_count` in struct `inode` of the Linux Kernel[3] is of type unsigned short, which can be overflowed by mapping one file more than 65535 times.

This is a bug, which was found in January 1998 is in the versions up to Linux 2.0.31

This is also a weakness in the design of the kernel. Again this emphasizes the fact that the design aspect of the operating system needs greater care. These problems would not have brewed up if more effort been put during the design phase

The obvious solution to this problem is to change the `i_count` variable to type unsigned long

B. Crashing the System by Eating Memory

This topic is related to previous one. The Linux memory manager allocates small chunks (64 bytes) of memory for every file mapped. By mapping a file many times, a process can eat all the available memory and actually stop the system responding even for the root processes.

C. Lilo Vulnerability

Linux uses lilo(Linux Loader) to load multiple operating systems. But the loader is highly insecure and very easy to break.

Lilo offers a lot of ways to get to the root account by people who have physical access to the machine. This is a very serious bug and this has not been patched even in the latest version of linux.

If some strange parameters are given at the boot prompt, Linux boots gracefully into the root account. The problem lies in the `inittab` file, which is used by INIT.

The design of kernel itself seems to be fallible. Network Operating Systems (NOS) should be designed with this view in

mind. The designers should meticulously plan the vulnerable aspects of operating system. The most blatant example of overlooking such vulnerabilities is strikingly evident in the example discussed above.

Ideally Linux should adopt a method similar to that used by FreeBSD[2]. LILO would then only load the kernel and the kernel itself would prompt for information and could be configured to not accept certain parameters

IV. FLAWS IN PROTOCOLS

The TCP/IP[4] protocol suite, which is very widely used today, was developed under the sponsorship of the Department of Defense (DOD). In spite of this, there are number of serious security flaws inherent in the protocols. Some include sequence number spoofing, routing attacks etc.

We discuss generic problems with the protocols themselves. As will be seen, careful implementation techniques can alleviate or prevent some of these problems

A. TCP Sequence Number Prediction

The normal TCP connection establishment sequence involves a 3-way handshake. The client selects and transmits an initial sequence number, the server acknowledges it and sends its own sequence number, and the client acknowledges the same. Following these three messages, data transmission may take place, i.e., for a conversation to take place, the client must first hear more or less a random number.

If, by any means, the random number can be predicted by an intruder then he can impersonate the trusted host thereby compromising many hosts in the network.

If the intruder were to perform this attack on a connection that allows remote command execution (e.g. the Berkeley rsh server), malicious commands could be executed.

In Berkeley systems, the initial sequence number variable is incremented by a constant amount once per second, and by half that amount each time a connection is initiated. Thus, if one initiates a legitimate connection and observes the random number used (say, through command like netstat), one can calculate, with a high degree of confidence, the sequence number which might be used on the next connection attempt.

Here the flaw lies in the protocol and we can use cryptographic algorithms (or devices) for a sequence number generation. The Data Encryption Standard (DES) is an attractive choice as the sequence number source, with a simple counter as input.

B. Routing Information Protocol Attacks

The Routing Information Protocol (RIP)[5] is used to propagate routing information on local networks. Typically, the information received is unchecked. This allows an intruder to send bogus routing information to a target host, and to each of the gateways along the way, to impersonate a particular host. The most likely attack of this sort would be to claim a route to a particular unused host, rather than to a network; this would cause all packets destined for that host to be sent to the intruder's machine (Diverting packets for an entire network

might be too noticeable; impersonating an idle work-station is comparatively risk-free). Once this is done, protocols that rely on address-based authentication are effectively compromised.

Defense against RIP attack can be done through a gateway one that filters packets based on source or destination addresses. This will block any form of host - spoofing (including TCP sequence number attacks), since the offending packets can never make it through.

C. The Internet Control Message Protocol

The Internet Control Message Protocol (ICMP) [6] is the basic network management tool of the TCP/IP protocol suite. The first, and most obvious target, is the ICMP Redirect message; it is used by gateways to advise hosts of better routes. As such it can often be abused in the same way that RIP can be.

Some of the flaws discussed above have been rectified in the recent versions. These flaws demand a greater insight into the intricacies of protocols

V. FLAWS IN THE IMPLEMENTATION OF PROTOCOLS

Though some protocols have been designed to be flawless their incorrect implementation invites hackers to invade the systems. A few implementations such as finger[7], ping services, DNS, FTP not reliable in spite of their protocols being secure.

A. The Finger Service

Many systems implement a finger service. This finger server displays useful information about users, such as their full names, phone numbers etc. Unfortunately, such data provides useful information to a password cracker. By running such a service, a system administrator is giving away data to hackers and crackers. Internet Worm is a good examples

B. Electronic Mail

Electronic Mail[8] is probably the most valuable service on the internet. Nevertheless, it is quite vulnerable to misuse. As normally implemented, the mail server provides no authentication mechanisms. This leaves the door wide open to faked messages. RFC 822 does support an Encrypted header line, but this is not widely used. (However, RFC 1040[9] proposes a new encryption standard for electronic mail).

C. The Post Office Protocol

The Post Office Protocol (POP)[10] allows a remote user to retrieve mail stored on a central server machine. Authentication is by means of a single command containing both the user name and the password. However, combining the two on a single command necessitates the use of conventional passwords, and such passwords are too vulnerable to password cracking.

D. The Domain Name System

The Domain Name System (DNS) provides a distributed database mapping host names to IP addresses. An intruder who interferes with the proper operations of the DNS can mount a variety of attacks, including denial of service and password collection.

E. The File Transfer Protocol

The File Transfer Protocol (FTP)[11], in itself, is not flawed. However, a few aspects of the implementation merit some care.

A problem area is anonymous FTP. While not required by the FTP specification, anonymous FTP is a treasured part of tradition of the internet. Nevertheless, it should be implemented with care. One part of the problem is the implementation technique chosen. Some implementations of FTP require creation of a partial replica of the directory tree; care must be taken to ensure that these files are not subject to compromise. Nor should they contain any sensitive information, such as encrypted passwords.

VI. FLAWS IN UTILITY PROGRAMS

People who are not in the design of operating systems generally write the utilities supplied along with the operating system. There are few inconsistencies, which creep in without the designer's knowledge (buffer overflows etc). These give rise to one of the most powerful attacks the *Stack-Smash*[12]

It is discussed in section VIII in detail.

A. Buffer Overflows

Over the last few months there has been a tremendous increase in buffer overflow vulnerabilities being both discovered and exploited. Examples of these are `syslog`, `sendmail 8.7.5`, `Linux/FreeBSD mount`, `Xt library`, `at`, etc.

A buffer is simply a contiguous block of computer memory that holds multiple instances of the same data type. C programmers normally associate with word buffer arrays. Most commonly, character arrays. Arrays, like all variables in C, can be declared either static or dynamic. Static variables are allocated at load time on the data segment. Dynamic variables are allocated at run time on the stack. Here we discuss the dynamic buffer overflows.

When a function is called all its parameters are pushed on a stack and the return address is stored. Then it branches to the function address. But if we overflow a buffer by overwriting the return address carefully, we can branch to any arbitrary address and execute the code we wish to. (For detailed explanation refer to section VIII).

If the utility programs fail to check the input, which is read in to the buffer, a clever intruder may set up the string causing the buffer to overflow, which results in executing his malicious code.

B. Vulnerabilities in utilities like compilers, loaders etc.

Compilers, Loaders, Linkers, Libraries form the heart of an operating system. Almost all programs use these utilities extensively. So if these utilities are flawed then there is no stoppage for the intruders.

One serious flaw is cited below.

1) *Standard Overflow in libc*: There lies a problem with `syslog()` and if some strange input is given it dumps core. We can make use of `su` to exploit this bug and gain root access. Bugs similar to these have been corrected in newer versions. To avoid these bugs, the designers have to be more careful in the implementation.

Designers should check programs using standard buffers and every effort must be made to check the bound of the input. Such inconsequential aspects are usually neglected and turn out to be serious security flaws. These should be checked for during the design and coding.

VII. COMPREHENSIVE DEFENSES

Thus far, we have described defenses against a variety of individual attacks. Several techniques are broad-spectrum defenses; they may be employed to guard against not only the attacks, but many others as well. Generalized techniques, which can be used to enhance System Security, are as below.

A. Security Tools

Many security tools are available on Internet to help system administrators. Some tools include `COPS`, `Chkacct`, `Crack`, `shadow`, `passwd+` etc.

1) *Computer Oracle and Password System*: `COPS` is a UNIX security status checker. It, essentially, checks various files and software configurations to see if they have been compromised (edited to plant a Trojan horse or Back door), and checks to see that files have the appropriate modes and permissions set to maintain the integrity of the security level.

Availability: `COPS` can be downloaded via anonymous FTP from cert.org

2) *Chkacct*: `Chkacct` is a `COPS` for the ordinary user. This tool is made available to the user to run, or it is run for them once per day. It will do an integrity check on the status of files in their own accounts and then mail them the results. This package makes users more aware of security controls.

Availability: `Chkacct` is distributed along with `COPS` package

3) *Crack*: `Crack` helps the security administrator identify weak passwords by checking for various weaknesses and attempting to decrypt them. If `Crack` can figure out a password, then the password must be changed to a better one. It is very likely that a determined intruder will be able to get the password too (using similar techniques, or the `Crack` program itself, since it is publicly available).

Availability: `Crack` can be downloaded via anonymous FTP from cert.org.

4) *Shadow*: The shadow password suite of programs replaces the normal password control mechanisms on your system to remove the encrypted password from the publicly readable file `/etc/passwd` and hides them in a place that only this program has permission to read. It consists of optional, configurable components, provides password aging to force users to change their passwords once in a while, adds enhanced syslog logging, and can allow users to set passwords up to a length of sixteen characters.

Availability: Shadow is available from USENET archive, which store the `comp.source.misc` newsgroup

5) *Passwd+*: *Passwd+* is a proactive password checker that replaces the `/bin/passwd`. It is rule-based and easily configurable. It prevents users from selecting a weak password so that program like "CRACK" can't guess it, and it provides enhanced syslog logging.

Availability: *Passwd+* (developed by Matt Bishop) is available via anonymous FTP from darthmouth.edu

B. Authentication

Many of the intrusions described above succeed only because the target host uses the IP source address for authentication, and assumes it to be genuine. Unfortunately, there are sufficiently many ways to spoof this address that such techniques are all but worthless. Some form of cryptographic authentication is needed. The data transmitted through the network should be encrypted and this makes intruder's tasks difficult.

C. Encryption

Suitable encryption can defend against most of the attacks outlined above. But encryption devices are expensive, often slow, hard to administer, and uncommon in the civilian sector. Link-level encryption encrypts each packet as it leaves the host computer and is an excellent method of guarding against disclosure of information. It also works well against physical intrusions; an attacker who tapped in to an Ethernet cable, for example, would not be able to inject spurious packets.

D. Firewalls

An Internet Firewall is a system or group of systems that establishes a security perimeter to control both incoming and outgoing network communication according to the Site's Security Policy. This helps in filtering the packets entering or leaving the network, thus making intruder's task more difficult.

VIII. THE STACK-SMASH METHOD

A. Introduction

As pointed out earlier, a buffer is simply a contiguous block of computer memory that holds multiple instances of the same data type. TO understand what stack buffers are we must first understand how a process is organized in memory. Processes are divided into three regions. Text, Data and Stack. We will concentrate on the stack region, but first a small overview of the other regions is presented.

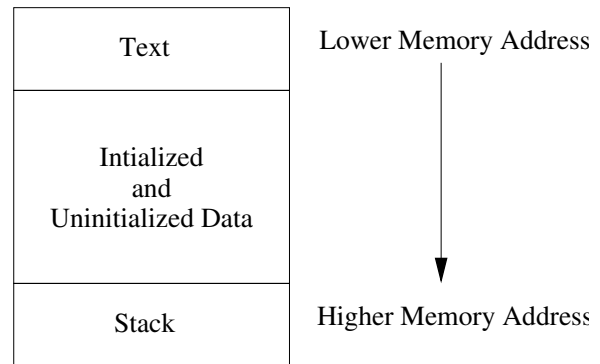


Fig. 1. Process Memory Regions

B. Process organization in memory

The text region is fixed by the program and includes code (instructions) and read-only data. This region corresponds to the text section of the executable file. This region is normally marked read-only and any attempt to write to it will result in a segmentation violation.

The data region contains initialized and uninitialized data. Static variables are stored in the region. The data region corresponds to the data-bss sections of the executable file. Its size can be changed with the `brk(2)` system call. If the expansion of the bss data or the user stack exhausts available memory, the process is blocked and is rescheduled to run again with a larger memory space. New memory is added between the data and stack segments.

C. The Stack Region

A stack is a contiguous block of memory containing data. A register called the stack pointer (SP) points to the top of the stack. The bottom of the stack is at a fixed address. The kernel dynamically adjusts its size at run time. The CPU implements instructions to PUSH onto and POP off the stack.

The stack consists of logical stack frames that are pushed when calling a function and popped when returning. A stack frame contains the parameters to a function, its local variables, and the data necessary to recover the previous stack frame, including the value of the instruction pointer at the time of the function call.

In addition to the stack pointer, which points to the top of the stack (lowest numerical address), it is often convenient to have a frame pointer (FP) which points to a fixed location within a frame. Some texts also refer to it as a local base pointer (LB). In principle, local variables could be referenced by giving their offsets from SP. However, as words are pushed onto the stack and popped from the stack, these offsets change. Although in some cases the compiler can keep track of the number of words on the stack and thus correct the offsets in some cases it cannot, and in all cases considerable administration is required. Furthermore, on some machines, such as Intel-based processors, accessing a variable at a known distance from SP requires multiple instructions.

Consequently, many compilers use a second register, FP, for referencing both local variables and parameters because their

```

void function(int a, int b, int c)
{
    char buffer1[5];
    char buffer2[10];
}

void main()
{
    function(1, 2, 3);
}

```

Fig. 2. Example1.c

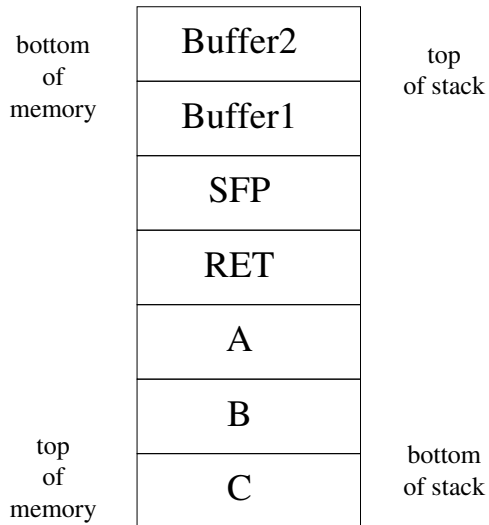


Fig. 3. Stack Organization

distances from FP do not change with PUSHes and POPs. On Intel CPUs, BP(EBP) is used for this purpose. Because the way our stack grows, actual parameters have positive offsets and local variables have negative offsets from FTP.

The first thing a procedure must do when called, is save the previous FPT (so it can be restored at procedure exit). Then it copies SP into FP to create the new FP, and advances SP to reserve space for the local variables. This code is called the procedure prolog. Upon procedure exit, the stack must be cleaned up again, something called the procedure epilog. The Intel ENTER and LEAVE instructions and the Motorola LINK and UNLINK instructions, have been provided to do most of the procedure prolog and epilog work efficiently.

Let us see what the stack looks like in a simple example 2:

After the function is called, the parameters of function are pushed on the stack and ret address and sftp are stored as shown in the figure 3. The local variables of function are next pushed on the stack.

D. Buffer Overflows

A buffer overflow is the result of stuffing more data into a buffer than it can handle. We write a program so that it overwrites the return address, and we can make it execute arbitrary code.

```

char shellcode[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0"
    "\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x56\x0c\xcd"
    "\x80\x31\xdb\x89\xd8\x40\xcd\x80"
    "\xe8\xde\xff\xff\xff/bin/sh";
char large_string[128];

void main()
{
    char buffer[96];
    int i;
    long *long_ptr = (long *)large_string;

    for(i = 0; i < 32; ++i)
        *(long_ptr + i) = (int)buffer;

    for(i = 0; i < strlen(shellcode); ++i)
        large_string[i] = shellcode[i];
    strcpy(buffer, large_string);
}

```

Fig. 4. overflow.c

E. Writing and Exploit

The code listing in figure 4 shows how to write an exploit. We filled the array large_string with the address of buffer, which contains the shell code. The shellcode is copied into the beginning of large_string array. The call to strcpy() copies large_string onto buffer without doing any bounds checking causing overflow. It overwrites the return address with the address where shell code is located. At the time of return from the function, a jump to shell code is done which in turn execs a shell.

F. Finding Buffer Overflows

As stated earlier, buffer overflows are the result of stuffing more information into a buffer than it is meant to hold. Since C does not have any built-in bounds checking, overflows often manifest themselves as writing past the end of a character array. The standard C library provides a number of functions for copying or appending strings, which perform no boundary checking. They include: strcat(), strcpy(), sprintf() and vsprintf(). These functions operate on null-terminated strings, and do not check for overflow of the receiving string. gets() is a function that reads a line from stdin into a buffer until either a terminating newline or EOF. It performs no checks for buffer overflows. If the target of any of these functions is a buffer of static size, and its other argument was somehow derived from user input there is a good possibility that it might be exploited as a buffer overflow.

IX. CONCLUSIONS

From the above discussions it is evident that considerable efforts have been put to make systems, which are inherently vulnerable, impervious to attacks. Nevertheless, the methods

cited above are quite efficient and are widely used. However, prominence must be given to make systems completely secure rather than trying to patch up the existing flaws. The flaws discussed above should be accounted for while designing a Network Operating System, Protocols and their implementations and Utilities. The thought process of the designers must be oriented to-towards security and system protection.

REFERENCES

- [1] A. S. Tanenbaum, *Distributed operating systems*. Englewood Cliffs, NJ 07632, USA: Prentice-Hall, 1995.
- [2] M. McKusick, K. Bostic, and M. Karels, "The design and implementation of the 4.4BSD operating system," *Addison-Wesley*, May 1996.
- [3] A. One, "i_count overflow security hole." [Online]. Available: http://www.insecure.org/spl0its/inode_count.html
- [4] J. Postel, "RFC 793: Transmission control protocol," Sept. 1981. [Online]. Available: <ftp://ftp.internic.net/rfc/rfc793.txt>
- [5] G. Malkin, "RFC 2453: RIP version 2," Nov. 1998. [Online]. Available: <ftp://ftp.internic.net/rfc/rfc1388.txt>
- [6] J. Postel, "RFC 792: Internet Control Message Protocol," Sept. 1981. [Online]. Available: <ftp://ftp.internic.net/rfc/rfc777.txt>
- [7] K. Harrenstien, "RFC 742: NAME/FINGER protocol," Dec. 1977. [Online]. Available: <ftp://ftp.internic.net/rfc/rfc1194.txt>
- [8] J. Postel, "RFC 821: Simple mail transfer protocol," Aug. 1982. [Online]. Available: <ftp://ftp.internic.net/rfc/rfc788.txt>
- [9] J. Linn, "RFC 1040: Privacy enhancement for Internet electronic mail: Part I: Message encipherment and authentication procedures," Jan. 1988. [Online]. Available: <ftp://ftp.internic.net/rfc/rfc1040.txt>
- [10] J. Myers and M. Rose, "RFC 1939: Post Office Protocol — version 3," May 1996. [Online]. Available: <ftp://ftp.internic.net/rfc/rfc1725.txt>
- [11] J. Postel and J. K. Reynolds, "RFC 959: File transfer protocol," Oct. 1985. [Online]. Available: <ftp://ftp.internic.net/rfc/rfc2228.txt>
- [12] Aleph One, "Smashing the stack for fun and profit," *Phrack Magazine*, vol. 7, no. 49, p. File 14, 1996. [Online]. Available: <http://www.phrack.org/show.php?p=49&a=14>