# Common Hacking Tools for Linux and Windows

CS 581 Semester Project
Spring 2002

Erin Fox
Jeremiah Bush
Sylvia Ashley
Ian Webb

## Abstract

For this project we tested and evaluated some of the hacking tools available in the public domain to discover their capabilities. Although we focused primarily on Linux tools, we did work with some Windows-based tools. We analyzed several tools from each of various categories and studied how the tools and methods work, and the vulnerabilities they exploit.

## Introduction

### Overview

With the widespread popularity of Internet technologies, there are bound to be people who abuse the system for fun or profit. A multitude of hacking tools are a click away for those who are interested, whether their motives are for offense or defense.

Research done by The HoneyNet Project, a group that studies hackers, found that a random computer connected to the Internet is scanned for vulnerabilities dozens of times daily. The time before a default installation of Red Hat 6.2 server is hacked is less than three days. A Windows 98 setup with file sharing enabled was attacked five times in the first four days. On average, systems undergo NetBIOS scans 17 times a day [1].

Clearly there is a need for computer systems large and small to be secure against such attacks. One way to discover an attacker's methods is to use the tools in the attacker's own toolkit. Many of these tools are available in the public domain, as they should be. Just as security through obscurity of a system's attributes doesn't work, neither does security through keeping the tools of the attacker secret. Chances are that anyone with malicious intent would be able to find the tools anyway through the

underground, and making the tools readily available to everyone allows security professionals to better defend against them.

The process of hacking a system typically involves four steps: reconnaissance, scanning, gaining access, and maintaining access. Reconnaissance involves gathering information on the target. Scanning is the process of trying to find a vulnerability in the target that can be used as an entryway into the system. Access is gained through weaknesses in applications and operating systems, and through sniffing, spoofing and session hijacking over local area networks. Maintaining access requires that the attacker cover his tracks to avoid being detected and install backdoors for future access. Once access is secured, the black hat can launch attacks from the compromised machine, or simply monitor it and steal information or CPU cycles. We evaluated tools from each of the steps following reconnaissance, concentrating primarily on tools used for scanning and gaining access.

**Lab Setup and Scope of Project**

The security lab provided for this class consisted of ten machines running Linux Red Hat 7.0 or 7.2, Linux Mandrake 8.1, and Windows NT and 2000, on a single static network. Most services were shut off except for ssh and telnet. There was no external access except via ftp or http, for known IP addresses only, since DNS was disabled. The tools we tested were either open source or were trial versions of commercial applications, and our project was limited to studying tools that attack Linux and Windows systems.

## Step 1: Reconnaissance

Reconnaissance, also called footprinting, involves finding out as much as possible about a target from information that is available in the public domain. An attacker can gain a wealth of valuable information by visiting an organization's Web site, researching the target on Web search engines, and monitoring newsgroups.

The domain name registrar that a target uses can be found by entering the organization's domain name in a search at *www.internic.net/whois.html* or *www.allwhois.com/home.html*, or one of the other whois databases. By contacting the target's registrar, the attacker can learn employee names and telephone numbers, email addresses, and addresses for the target's DNS server [2].

Due to the structure of this project, where the hackers are familiar with the targets and use the same network, these tactics were unnecessary. A comprehensive study of reconnaissance techniques would also cover such low-technology tactics as social engineering, physical break-ins, and dumpster diving, but for the same reasons, these are not within the scope of this project.

## Step 2: Scanning

**Network mapping**

Network mapping is the process of discovering information about the network topology, including gateways, routers, and important servers. The first step is to find live systems, given a range of addresses

in the target network. This process is called sweeping. To find live hosts, hackers ping them by sending ICMP packets, and if a machine is up it will send an ICMP echo reply. ICMP messages can be blocked, so an alternative is to send a TCP or UDP packet to a port such as 80 (http) that is frequently open, and live machines will send a SYN-ACK packet in response. Once the live systems are known, tools such as the standard Unix utility traceroute can provide additional information about the network topology by discovering the paths taken by packets to each host, which provides information about the routers and gateways in the network and the general network layout.

**Port scanning**

Once the IP address of a target system is known, an attacker can then begin the process of port scanning, looking for holes in the system through which the attacker can gain access. A system has $2^{16}$-1 port numbers, and one TCP port and one UDP port for each number. Each of these is a potential entryway into the system. If a port is open, there is a service listening on it; well-known services have assigned port numbers, such as http on TCP port 80 or telnet on TCP port 23. Port scanning is the process of sending packets to each port on a system to find out which ones are open. An attacker may focus on a single port or the entire range, but the more ports scanned, and the faster the packets come, the more obvious it will be to the system being scanned.

The most basic port scanners, such as portscan.c or netcat, will scan a range of TCP (and sometimes UDP) ports using the simplest methods. These scanners do a "polite" scan that follows the TCP three-way handshake protocol: the attacker sends a packet with the SYN code bit set, the service will respond with a SYN-ACK if open, and the attacker completes the connection with an ACK. This fully opened connection is likely to be logged at the server side, making this an easy scan to detect.

UDP scans are more simplistic; if an ICMP port unreachable packet is received in response to a UDP packet, the port is closed, but as UDP packets are inherently unreliable, no response only means the port is probably open.

If an attacker wishes to be more stealthy, there are more advanced scanners available such as nmap that are capable of a wider variety of TCP scans that are harder to detect. Nmap allows an option for a TCP SYN stealth scan in which the third message is not an ACK but a FIN that forces the TCP connection to be closed before fully opening. This half-open connection is not logged at the target but may be noticed by routers or firewalls that record the original SYN packet. Nmap can also run other TCP scans that violate the TCP protocol, such as Xmas tree and null scans. These typically work only on non-Windows machines and involve sending unexpected packets; if a RESET is received, the port is closed, while no response means it is open.

Nmap also allows options that give the attacker more control over the packets sent. The attacker can set the rate at which packets are sent, since changing the timing to space out the packets can help avoid raising the target's suspicions that it is being scanned. If the rate is set too fast, packets can be lost and incorrect results will be returned. The attacker can also fragment the packets to avoid intrusion detection systems, many of which only look for the whole suspicious packet to be sent at once. Nmap even allows the attacker to set the source port, for example to 80 to appear as web traffic to a packet filter, as well as to set a decoy source address to obscure the real address by sending an extra packet per decoy address.

Additional information about the system can be gained during the port scanning process. Using TCP stack fingerprinting, nmap can determine the operating system type of the scanned machine. This method involves sending different illegal combinations of TCP code bits, to which different operating systems

will return different responses.

**Vulnerability scanning**

One essential type of tool for any attacker or defender is the vulnerability scanner. These tools allow the attacker to connect to a target system and check for such vulnerabilities as configuration errors, default configuration settings that allow attackers access, and the most recently reported system vulnerabilities. Most commercial NSSs are expensive and do not come with the source code, while the open source NSSs are free and the source code is readily available. We focus on the open-source tool Nessus.

Nessus is an extremely powerful network scanner. Hacking Linux Exposed claims that it is probably the most up-to-date scanner currently available [3]. Nessus can be configured to run a variety of attacks. Users can also turn off the dangerous attacks that might bring a computer down (such as Winnuke when scanning a Windows NT machine).

Nessus includes a variety of plug-ins that can be enabled depending on the type of security checks the user wishes to perform. These plug-ins work cooperatively. Each test can specify what is needed to proceed with the test. For example, if a certain test requires a remote ftp server and a previous test showed that none exists, that test will not be performed. This allows the tests to run faster as the logic does not have to be built into each test. Not performing futile tests also speeds up the scanning process [3]. Plug-ins are updated daily and are available from the nessus website.

Nessus includes its own scripting language, called Nessus Attack Scripting Language (NASL), which can be used to create individualized attacks and incorporate them with the other plug-ins. Although attacks could be written in C, Perl, Python, or a variety of other languages, NASL was designed to be an attack language.

When attacking multiple machines, Nessus attempts to run attacks in parallel to save time. Attacking four machines seems to take about the same amount of time as attacking one machine. Although it takes a relatively long time to run, it is worth the wait. The output is incredibly detailed and there are multiple formats available for the reports. The reports give information about security holes, warnings, and notes. Nessus does not attempt to fix any security holes that it finds. It reports them and gives suggestions on how to make the vulnerable machine(s) more secure.

We ran Nessus multiple times in the security lab. On our first run, we attacked a machine running Windows NT. Because we had all of the dangerous attacks enabled, we were able to "blue screen" the computer with the Winnuke attack. Next, we ran Nessus against four of the machines in the security lab that were running Mandrake Linux 8.1, Red Hat Linux 7.1, Red Hat Linux 7.1, and Windows 2000. Between the four machines, Nessus found 11 security holes, 44 security warnings, and 27 security notes.

## Step 3: Gaining access

Once the vulnerabilities of a system have been discovered, an attacker can often find an exploit for one of them on the web. Many exploit databases are searchable by operating system type and version and by exploit type. One common type of exploit is the stack-based buffer overflow attack.

**Stack-based buffer overflow attacks**

Stack-based buffer overflow attacks take advantage of poorly written applications and operating systems. They are a commonly used exploit that can enable an attacker to execute commands on the target machine.

One example of code that allows a buffer overflow is a program with a function that tries to copy a string made up of 256 of the character "A" into a local variable with memory allocated for only 16 characters. If the string length is not verified, the string overflows the memory location, the return address will be outside the process address space, and a segmentation fault is returned. But if the attacker can overwrite the return pointer so that it points back into the buffer, he could direct execution flow back into the buffer, where he has placed the machine code he wants to execute [4].

These attacks must be adapted to the particular processor and operating system in use, because the raw machine code is processor-dependent, and different operating systems handle command execution differently.

To find programs vulnerable to stack-based buffer overflows, an attacker can search the source code or use a debugger on the program. If the source code is not available, the exploit creator can execute the program on his own machine using huge amounts of repeating data for each possible input of the program. If he can make the program crash using, for instance, thousands of the character "A," he can search the processor's registers after the crash for the instruction pointer value. If it contains the hex value of "A," he can be fairly sure that a buffer overflow occurred [2].

When one of these poorly written function calls is found, the exploit creator must analyze it using test inputs until he knows enough to write code that will correctly push instructions onto the stack and overwrite the return pointer. Formatting everything correctly can be rather tricky. To increase the chance that the return pointer will point to the beginning of the attacker's code, the attacker will often use a series of "no operation," or NOP instructions, before his code. This way, the pointer can reference any in the series of NOPs. When program execution comes to the NOP, the processor does nothing, and execution is passed to the next NOP, in what is called a "NOP sled." This continues until eventually the first instruction of the attacker's code is reached and executed [2].

An attacker will usually want to direct execution to instructions to spawn a command shell, using the *execve* system call. From this shell he can issue any other command.

This all points to the importance of good programming practices such as bounds checking for C programs. It is also important to check library functions you use that are written by another programmer. To detect stack-based buffer overflow exploits used against a target, an intrusion detection system can monitor the network for such signs as NOP sleds, typical machine language code used in these exploits, or frequently used return pointers.

**Password Attacks**

The most common security tool used is the password. A weak password can be the weakest link in a system's security, allowing access to some of the most sensitive secrets of an organization.

Because creating the password is usually left up to the user, who may not know or care how to make a more secure password, some passwords are easily guessed. In addition, many systems allow access through default passwords which, if not removed by the system administrator, can be easily found and used for unauthorized access.

One method to find weak passwords is to write a script that repeatedly tries to log in to a machine

using a common user name, or one that the attacker already knows about, and tries all words in a dictionary until one matches and the attacker has logged in. This is a very slow process and is subject to detection by a system administrator. If the system is set up to allow the user only a certain number of password tries, then the attacker will not be able to make more than that number of guesses until he is locked out.

Typically, system passwords are stored as encrypted or hashed representations. When a user logs onto the system with his password, it undergoes the same transformation as the passwords in the password file and is compared with the user's encrypted password value. If the values match, the user is allowed to login.

Obviously, an attacker must have access to the password file in the first place in order to crack those passwords. The first problem on many systems is that, since the password file (*/etc/passwd* in Linux systems) is required to be readable by every user on the system, the attacker can get a copy as soon as he has gotten any kind of access via a normal user. The attacker can also use a buffer overflow exploit (although this must be of a root suid program if the passwords are shadowed). In addition, if an attacker can cause a program that reads the password file to crash, it may create a core dump, and the core file can be searched for user names and encrypted passwords.

After the attacker steals the file of encrypted passwords, he uses a cracking tool to try to recover the plaintext password. Password cracking tools follow three steps that are repeated until a match is found. They guess a password, then encrypt it, and finally compare it to the user's encrypted password from the file. If it matches, the password has been cracked. If no match occurs, the process is repeated. Typically, all words in a dictionary file are used as the password guesses. The work of guessing, encrypting, and comparing passwords can be distributed among several different computers to speed up the process.

Not all password cracking utilities work strictly from a dictionary file. Many create password guesses based on information about the user such as their user name or other known user information. They may also generate potential passwords based on rules that are applied to dictionary terms or to user information. Rules for constructing password guesses include the following, among many others: straight dictionary words; combinations and permutations of dictionary words, such as the same word concatenated to itself, words spelled backward or with various letters capitalized, or with numbers or letters appended or prepended; passwords cracked previously; words from a themed dictionary (such as Star Trek, Shakespeare, or Monty Python terms, sports teams, or famous people's names); or brute-force attacks. Brute-force attacks take the most time and are generally not run against each user in a password file but on a single user (such as root) that would provide the most benefit if cracked.

This is not the only method of obtaining user passwords; sniffing and keystroke logging are also used for this purpose. In addition, passwords are not always necessary in order to gain access to a system, even as root, if the attacker has a good exploit.

**Password Cracking Tools**

Not all password cracking tools are created equal. They range from utilities that simply try each word in a dictionary, to those that try every possible combination of letters, numbers, and/or punctuation, to some that will try hundreds of permutations on each dictionary word before resorting to brute force. In addition, run parameters for the tools are sometimes configurable. Some tools, such as Crack, may be set to do password attacks across a network, to email users that their passwords have been cracked, to prioritize various groups of dictionary (or other) words, or to run only when the system load is light.

Several tools come with additional utilities to merge the */etc/passwd* file with the actual shadowed passwords, since most tools require a traditional password file format.

**Password Cracking Tool: Crack**
***(http://packetstorm.linuxsecurity.com/Crackers/crack/)***

Crack is a popular password cracker with many configurable rules. However, it does not run on current Linux systems, as the newest versions of Linux use a newer method of encrypting the stored passwords.

**Password Cracking Tool: John the Ripper *(http://www.openwall.com/john/)***

John the Ripper required several source code modifications to run on the security lab machines using Red Hat Linux. It also needed changes to the Makefile.

Cracking times, using the default dictionaries that come with the Linux system, are as follows:

User erin3 with password erin3erin3 took less than one second.
User erin2 with password 12345678 took less than 10 seconds.
User erin6 with password !@#$%^&* took somewhere under two minutes.
User erin1 with password bogus took about five minutes.

John the Ripper requires the user to have a copy of the password file.

**Password Cracking Tool: XCrack *(http://packetstorm.linuxsecurity.com/Crackers/)***

XCrack doesn't do much with rules. It will find any passwords that match words in the dictionary file the user provides, but it won't aply any combinations or modifications of those words.

**Password Cracking Tool: LC3 *(http://www.atstake.com/)***

LC3 is the latest version of L0phtCrack. It is a commercial product, for use on Windows, available for $249 as of April 8, 2002. Earlier versions of L0phtCrack are available at no cost.

LC3 can obtain encrypted passwords through several methods. Many of these methods require administrator access to the machine that the passwords are being retrieved from. Once the encrypted passwords are retrieved, LC3 has four methods of cracking them: User Info Check, Dictionary, Hybrid and Brute Force. The trial version we used does not have the Brute Force method of password cracking available. User Info Check will find all passwords where the user name is the same as that user's password. The Dictionary method is used next. LC3 systematically goes through its dictionary of 250,000 words and tries every word in an effort to find the password. Next, the Hybrid attack is used. This attack uses the same dictionary, but appends non-letter characters to each dictionary word.

As a last-ditch-effort, LC3 resorts to the Brute Force method. This method will first try all potential one-digit passwords, then all two-digits passwords, and so on until the password is found [5]. This method can be configured to use all 26 letters, the digits 0 through 9, and 32 special characters such as > # % & / or any subsets or combinations of those groups. This method may take an extremely long time to crack passwords.

Common passwords that use the letters A-Z and the numbers 0-9 can usually be cracked in about a day. It can take up to a hundred days to crack more complex passwords that use characters such as #_}* [6].

The most notable limitation of LC3 is that it cracks only passwords on Windows machines. Another shortcoming is the very limited dictionary. This can be remedied by installing another dictionary, although using a much larger dictionary slowed down both the dictionary and hybrid methods.

We set up different user accounts on security2 just to run LC3 and see how good it was at obtaining passwords. One of our users had a login name of "guest" and the password "user". LC3 was unable to crack this password using the first three methods. The default dictionary contained "user1" but not "user".

Another user we set up had a user name of "smithers" and a password of "burns". This time the hybrid method had to be used to find the password. This is because the dictionary contained "burn", "burned", and "burning", but not "burns".

We also set up a user that had a very common curse word as a password. LC3 never found this password with the methods we had available. It seems that the creators of this program were too concerned about not offending anyone to include socially unacceptable yet common words in the dictionary.

**Sniffing**

The technique of sniffing involves obtaining information from the local area network such as files, email, user names or passwords. In order to run a sniffer program, the attacker must have access to an account on the target system. After gaining access to this one account (through a buffer overflow attack, cracked password or by other means), the attacker can launch an "island-hopping" attack. The attacker installs the sniffer on the first machine and collects information about traffic on the same LAN segment. As user names and passwords to other machines are captured, sniffers are installed on them, and even more machines can be captured.

**Sniffing tool: LC3** ***(http://www.atstake.com/)***

When running LC3, the "Get Encrypted Passwords" menu states that Sniffing captures encrypted hashes in transit over your network. Logins, file sharing and print sharing all use network authentication that can be captured. This option does not require administrator access.

**Sniffing tool: Hunt** ***(http://lin.fsid.cvut.cz/~kra/index.html)***

Hunt is easy to install. It has a nice menu interface that allows the user to select a currently open connection, which he can listen in on or reset. The problem with Hunt is that when you select a connection, it has to be already open, so usually this means the user name and password have been transferred already and won't be caught by subsequent sniffing.

**Sniffing tool: SuperSniffer (*http://dhp.com/~ajax/projects/*)**

Running SuperSniffer captures a telnet session from one machine to another that logs in as root, runs the command "w" to see who is doing what on the system, and then logs off. The user name and password are fairly obvious in the output.

**Sniffing tool: LinuxSniffer (*http://www.hackpalace.com/usa/*)**

We started LinuxSniffer on one machine and logged a connection between two others. The output of LinuxSniffer lists each character transmitted; the user name and password

characters are in red. The output is not easy to read, but it would be simple to wrap a script around this to extract only user name and password information, or to figure out all the commands run from the telnet session.

**IP address spoofing**

When an attacker wants to hide the IP address of the computer from which he is attacking, he carries out IP address spoofing to fool the target server into believing contact is being made from a source other than the attacker. In this way the attacker can pretend to be a trusted contact to override applications that depend on IP addresses for authentication. The attacker may just want to hide his real identity so his actions can't be traced back to him.

The simplest way for an attacker to spoof an IP address is to change his address to that of another system using the Unix *ifconfig* command. Alternatively, the attacker could use nmap (described above under port scanning) to generate packets with the fake IP address in their headers. In this scenario the target will send any response packets to the spoofed address, so its usefulness is limited to situations where the attacker needs to obscure the source of packets, such as in a denial-of-service attack [2].

For more sophisticated attacks that require interaction between the attacker and target machines, IP address spoofing can exploit vulnerabilities in the Unix r-commands such as rlogin and rsh. If, for instance, a user wants to be able to move easily between two machines upon which he has an account, he can set up a trust relationship by entering the user name of each in the other's *.rhosts* file. Then the user can use any of the r-commands without needing to supply a password. These commands allow the trusted system's IP address to authenticate the user [7]. When an attacker replaces his actual IP address with that of a trusted system, he can log in to the trusting system without supplying a password.

**Session hijacking**

Using a combination of sniffing and spoofing techniques, session hijacking tools allow an attacker to steal an established login session. With a successful session hijacking, the victim's login session vanishes and he usually attributes it to network problems and logs in again.

After a source machine establishes a session with a destination machine, an interloper can launch host-based session hijacking if he has super-user access on either the source or destination machine. On a Unix system, the attacker can use a tool to interact with local terminal devices, or ttys, that are used in telnet and rlogin sessions. If the attacker has root, he can read all the session data from the target's tty and place keystrokes into the tty.

If the attacker does not have access to an account on either of the machines between which he wants to steal a session, he must use a network-based session-hijacking technique. This type of session-hijacking tool uses a sniffing technique on a segment of the network carrying traffic passing from the source to the destination to monitor the packets and their TCP sequence numbers. When the attacker decides to hijack the session, he inserts traffic into the network with the source IP address of the actual source instead of his own, placing the correct TCP sequence numbers on the packets. The destination machine thinks the traffic came from the legitimate source and follows the commands. The attacker has hijacked the session.

These session hijackings are successful even with the use of strong authentication, as long as the conversation itself is not encrypted. After the initial authentication with a time-based token, for instance, the ensuing session sent in plaintext can be hijacked [2].

**Session-hijacking tool: Hunt** ***(http://lin.fsid.cvut.cz/~kra/index.html)***

One of Hunt's advantages over other session-hijacking tools is that it uses techniques to avoid ACK storms. Recall the TCP three-way handshake protocol and imagine what happens when an attacker injects traffic into this stream. The source machine Alice opens the TCP connection by sending a packet to the destination machine Bob with the SYN code bit set and with her initial sequence number (ISN). Bob responds to Alice with a packet with the SYN and ACK code bits set and his ISN. Alice completes the handshake by sending Bob a packet with the ACK bit set. Each subsequent packet Alice sends to Bob will have an incrementally higher sequence number than her ISN, and each subsequent packet Bob sends to Alice will have an incrementally higher sequence number than his ISN, as well as a number acknowledging the sequence numbers of the packets received so far. In this way they can synchronize the receiving of each other's packets [2].

Now, if the attacker sends Bob a packet with Alice's IP address then Bob's acknowledging packet will contain a sequence number that is higher than that of the packet Alice sent last. She responds by resending her last ACK packet in an attempt to resynchronize the connection. As the attacker continues sending packets to Bob, Alice continues getting out-of-order ACK packets from Bob and continues to send Bob her last ACK packet. This ACK storm will eventually cause the connection to be canceled.

Hunt avoids this ACK storm and the dropping of the connection by using ARP spoofing to establish the attacker's machine as a relay between Alice and Bob. When a system prepares to send a packet over a LAN, it first sends out an Address Resolution (ARP) query to all the other systems on the LAN asking which of them has the Medium Access Control (MAC) address that corresponds to the IP address in the packet's header. The destination system replies with its MAC address, which the source system stores in its ARP cache for a certain period of time. For that period, the source system uses the data from its ARP cache to send transmissions to that destination. The attacker subverts this process by sending an unsolicited ARP response to Alice that maps Bob's IP address to a fake MAC address, and by sending an unsolicited ARP response to Bob that maps Alice's IP address to a fake MAC address. Both Bob's and Alice's systems overwrite these fake MAC addresses into their ARP caches, so that the packets they send to each other will go to fake addresses. They now cannot send packets to each other for the lifetime of the ARP cache [2].

Now the attacker uses Hunt to sniff the packets Alice and Bob send over this connection. The attacker can choose to act as a relay and forward these packets to their intended destinations, or he can hijack the session. The attacker can type in commands that are forwarded to Bob but which Alice can't see. Any commands Alice types in can be seen on the attacker's screen, but they are not sent to Bob. Then Hunt allows the attacker to restore the connection back to Alice when he is done with it.

Hunt can handle both LAN connections and connections between a user and the Internet. We used Hunt to hijack a telnet session over the security lab network.

## Step 4: Maintaining access

**Covering tracks**

The first step in maintaining access is to make sure the system administrators don't realize there has been a compromise. Attackers must hide their presence on a system by removing evidence of events associated with establishing access, changing privileges, and installing rootkits and backdoors [2]. This includes the evidence of such actions as stopped and restarted services, file access and update times, and failed logins.

Deleting entire log files hides these events but the complete disappearance of an important system file can indicate to a system administrator that an attack has occurred. The attacker should instead edit files line by line. Attackers especially need to alter files that show logins, including wtmp, utmp, and lastlog.

Because utmp and wtmp are in binary format, special editors are needed to remove entries from these files. A number of these editors are available online, and one version or another is frequently packaged into rootkit distributions as one of the most basic tools for hiding evidence of an attacker's presence on a system. However, care must be taken to find an editor that runs on the correct version of Unix, and to check the source code for the correct log file locations, to prevent errors that might be noticed in other log files.

These editing tools vary in their capabilities and sophistication. The most basic editors work on only one of these files and have little real intelligence when it comes to removing entries. In this category, utmp.c is a program that operates only on the utmp file, and it removes all entries for a specific user. In a similar fashion, wtmped operates only on wtmp, removing all evidence of logins by a particular user. This indiscriminate erasing of entries can raise an administrator's suspicions. Obviously, if the system shows that root, or another user who uses the system regularly, has never logged in or is not currently logged in even though a login session is open, it is fairly easy to deduce that the system has been compromised. The same is true if the output of "who" does not match up with the output of "last," which happens when only one of the login log files is edited.

At the other end of the spectrum is wipe, a program that operates on wtmp, utmp, and lastlog. It will remove the entries for a specific user on a specific terminal, allowing more control over the erased entries. Zap2 also works on all three files, removing only the most recent entry for the specified user from each file. This makes the changes less obvious to a system administrator, since entries unrelated to the attack are unaffected.

These programs change only the file contents; they do not alter the last access times of the files, which may tip off an administrator if they were written after the time of the last login. In addition, because the log files themselves are owned by root, these editors must be run as root.

**Backdoors and Trojan Horses**

In general, Trojan horses are programs that communicate information from one "infected" system to another system. For a Trojan horse to be useful, a user must install the program, or be tricked into installing it. Back Orifice and SubSeven are a type of Trojan horse called a backdoor. A backdoor allows a hacker to take over a victim's computer. In the case of the SubSeven backdoor there are only a few things that a remote user can't do to the victim's system, one of which is to turn on the computer remotely.

**Trojan horses: SubSeven and Back Orifice (*http://sub7crew.org* and**

***http://bo2k.sourceforge.net)***

These are termed "remote administration tools" by their writers, while they are commonly known as Trojan horses to anti-virus programs and normal users. Back Orifice is capable of taking over a Win2000 computer, while SubSeven seems to most easily work on Win98 computers.

These two tools have several things in common. They both come with two pieces of software, the client and the server. The server is the piece that the "remote administrator" will use to infect the victim's computer. The client is the piece that the attacker will use to monitor the victim's computer. Both programs allow for complete access to the victim's files. The hacker can copy, move, rename, delete and even change any file or folder in the victim's computer. Both have keylogging and password retrieval capabilities as well.

Back Orifice seems to be more of a serious hacker's tool. It doesn't have some of the frills the SubSeven has. It appears to be a tool that will give the attacker more information about other machines on the victim's network, therefore allowing the hacker to gain deeper access to the victim's network.

SubSeven, on the other hand, is geared toward beginners. When taking over a victim's computer, not only can the attacker control the victim's files, but he can control the entire system. It allows an attacker to change the appearance of the victim's screen, open and close the CD drive remotely, turn off the start button, hide the taskbar, turn off the monitor and even turn off the Ctrl-Alt-Delete button combination.

Possibly the most dangerous capability of the SubSeven, however, cannot be demonstrated in our security lab. It allows the attacker to use the victim's computer to participate in distributed denial-of-service attacks without the victim's knowledge. This allows the attacker the ability to cripple a network while not exposing himself to detection.

**Backdoor tool: netcat** ***( http://www.lopht.com/research/tools/)***

Netcat is a multipurpose networking tool capable of a variety of functions from port scanning and opening connections to remote ports, to creating backdoor shells for root access. It runs in either client mode or listening (server) mode - taking data from stdin and printing to stdout. In client mode, all data from standard input is transferred directly to whichever port netcat has been directed to open; in listening mode, it will open any unused port on the local machine, wait for a connection, and send to standard output any data it receives.

To create backdoor command shells, the attacker must deliberately compile netcat with a known security hole, adding the line #define GAPING_SECURITY_HOLE to the netcat source. With this value set, netcat can run a command (such as /bin/sh) during a connection. The attacker must already have root access to the victim's machine; this process is meant to help preserve that access. For a passive backdoor shell, the attacker can set up a listener on the victim's machine (as root) on port 3000 that runs a shell "nc -l -p 3000 -e /bin/sh". Then on the attacker's own machine, the attacker runs "nc victim_name 3000" to get root shell access at any time. To push a backdoor command shell, the attacker sets up a listener on his own machine. On the victim's machine, he connects to the attacker's machine and pushes a command shell using the "-e /bin/sh" flag as before. In

either case, the victim's port must be reachable from the outside for this to work, however routers or firewalls may interfere with this scheme.

Netcat is also capable of relaying traffic along a chain of systems running netcat listeners and clients. An attacker can set up netcat clients and listeners to forward traffic such that the attacker's location is hidden. If the attacker controls different machines, he can set up listeners on each one to direct any packets it receives to a client on the same system, which then passes the packets on to the next machine in the chain. The victim must trace the attack back across all these machines to find the original attacker. If relays cross language and political boundaries, the investigation will be significantly hampered [2]. One way to do this is via the Unix mknod command to create a special file used as a pipe. Command is "mknod myfile p" (p indicates is FIFO). Then run netcat as in "nc -l -p listen_port 0<myfile | nc next_machine next_port |>myfile". The backpipe file "myfile" is used to direct response traffic from the destination to the source.

**Keystroke loggers**

A keystroke logger, or keylogger, is a Trojan horse that captures keystrokes and stores them in a hidden file. Different implementations of keyloggers include hardware versions (such as trojaned keyboards) and software versions. Although there are many keystroke loggers available for use on Windows machines, very few reliable keyloggers are made for Linux. We tested Invisible Keylogger 97 (ik97) on a Windows 2000 machine. This particular keylogger is a software version that runs as a background process.

**Keystroke logger tool: Invisible Keylogger 97** ***(http://www.amecisco.com/index.htm)***

After installing ik97, the user must reboot the computer. This will automatically start the program. The program stores the keystrokes in a binary file. Running the included viewer program creates a human-readable file.

The shareware version of ik97 is limited to capturing 500 keystrokes; after that the program will exit without warning. The shareware version also starts itself after the Windows login and password dialog. Thus, when using the shareware version, we were unable to capture any login names or passwords. According to the readme file, the commercial version of ik97 will start when the Windows login and password dialog box appears. The commercial version also has no limits on the number of keystrokes captured.

We used ik97 to capture a file created with Borland JBuilder 4. In JBuilder, we started a new project. The information we entered included the project name, author, company, and things to do (on the project). The keyboard logger worked as promised. Although we did not try every key or every combination of keys, it did capture all the keystrokes typed.

**Rootkits**

One way an intruder can maintain access to a compromised system is by installing a rootkit. A rootkit contains a set of tools and replacement executables for many of the operating system's critical components, used to hide evidence of the attacker's presence and to give the attacker backdoor access to the system. Rootkits require root access to install, but once set up, the attacker can get root access back at any time.

The executables included in rootkits contain back doors or functionality that tends to overlook hacker activity. For example, rootkits typically include a version of the login program that allows the attacker to log in as root with a password the attacker knows but that is independent of the real root password, so that if the root password changes, the attacker's access is unaffected. Such a login program will also not write the login to the usual utmp and wtmp files. However, the backdoor login program must mimic the original as closely as possible, especially in terms of error messages and prompts, to avoid detection.

Other common modified programs include *ps*, which will not show an attacker's processes; *ls* and *find*, which hide an attacker's files; *ifconfig*, which hides network interfaces in promiscuous mode (indicating a sniffer); and *netstat*, which hides ports where an attacker's process is listening [2]. One Linux rootkit, lrk5, contains replacement executables for such programs as *du*, *chsh*, *top*, *su*, *inetd*, *sshd*, *syslogd*, *tcpd*, and many others.

In addition to the system programs, there are often other tools included in the rootkit. There may be a sniffer, which collects passwords onto other systems sent via telnet, login, or ftp sessions. The rootkit may also have a utility to reset the creation, modification, and last access times for any replaced programs, and pad the executables so that their checksums are the same as the originals. Rootkits also usually have a log file editor to erase their presence in the wtmp and utmp logs.

Beyond this, there are some rootkits that utilize loadable kernel modules to modify the operating system itself. This kind of rootkit is capable of just about anything, including everything in the traditional rootkits plus hiding of files, processes, networks, and kernel modules, as well as execution redirection, in which the operating system can run a different program in place of a requested one.

## Conclusions

Most problems we encountered with this project stemmed from the setup of the security lab. Because the machines were installed with a recent version of Red Hat, many security holes in earlier versions had been fixed. Also, because almost all services were turned off, there were very few ways of entering the system legitimately, much less illegitimately. The machines were configured in a very secure manner, giving us few opportunities to break in.

We also found some of the tools difficult to install on the lab machines, because we needed to make changes to source code to get the programs to compile. Some of the CD-ROMs that came with our books did not contain the most current versions of the tools; when we tried to install them on the security lab machines that run the newest versions of Linux, we ran into problems.

In addition, the scope of our project was too wide and shallow. Until we began investigating, we weren't aware of the number and variety of hacking tools available. Each tool targets one or more of a long list of vulnerabilities. Understanding how these tools work required additional research on TCP/IP protocols and networks. We attempted to cover one or more tools in each category but found this came at the expense of a less thorough examination and testing of each.

It is inevitable that hacking tools will get into the hands of black hats. Many of the tools available in the public domain are so easy to use that they require no understanding of the underlying principles that make them work. These factors lead us to strongly support a policy of complete disclosure.

It is imperative for security administrators to be familiar with the current crop of tools and to test them on their systems to discover their vulnerabilities before a hacker does. Open source programs can be

checked for Trojan horses or other such malicious code, which should be reported immediately. As soon as a new exploit is discovered, a patch to close the vulnerability can quickly be developed and distributed. There is really no difference between hacking tools and counterhacking tools. They are different only in the way they are used.

## Appendix 1 Terms and Definitions

**backdoor**
a program that bypasses security controls to let an attacker gain and regain access to a computer

**black hats**
those who attack other computer systems with malicious intent

**exploit**
a program or method an attacker uses to take advantage of a vulnerability on a computer system

**firewall**
a system that monitors network traffic and lets some through while blocking other traffic

**footprinting**
another name for the reconnaissance step; refers to the process of scoping out the target

**honeypot**
a decoy system set up by a security administrator used to attract black hat activity for the purpose of studying the attacker's methods or for diverting unwanted attention away from the actual system

**IDS (intrusion detection system)**
a system that looks for and reports black hat activity to the system administrator

**keylogger**
also known as a keystroke logger, this tool intercepts and records each keystroke made on a target keyboard

**packet filter**
network software that can transmit packets into or out of a network based on the header information

**rootkit**
a tool that modifies the operating system software to allow backdoor superuser access by an attacker

**scanning**
determining which target systems are alive and reachable from the Internet; includes port scanning, OS detection, vulnerability scanning

**session hijacking**
the technique in which an attacker steals a connection already established between a source and destination

**shadowing**
moving encrypted passwords out of world-readable */etc/password* and into a new file accessible only by root

**sniffing**
: the process of capturing and analyzing network traffic

**spoofing**
: any technique use to impersonate another computer user

**TCP stack fingerprinting**
: a method used to identify the target machine's operating system by analyzing the response from sending illegal TCP packets

**Trojan horse**
: a program that appear to or does carry out a particular function but which secretly performs other tasks

**white hats**
: the computer security professionals who attack a particular system in order to find and fix vulnerabilities

**worm**
: a malicious program that spreads copies of itself through a network

## Appendix 2 Tools Investigated

| | |
|---|---|
| netcat | http://www.lopht.com/research/tools/ |
| nessus | http://www.nessus.org |
| xcrack | http://packetstorm.linuxsecurity.com/Crackers/ |
| nmap | http://www.insecure.org/nmap/ |
| portscan.c | http://www.ryanspc.com |
| probe_tcp_ports.c, probe_udp_ports.c | http://www.ryanspc.com |
| Crack | http://packetstorm.linuxsecurity.com/Crackers/crack/ |
| Hunt | http://lin.fsid.cvut.cz/~kra/index.html |
| Sub7 | http://sub7crew.org |
| Invisible Keylogger97 | http://www.amecisco.com/index.htm |
| BackOrifice | http://bo2k.sourceforge.net |
| SuperSniffer | http://dhp.com/~ajax/projects/ |
| LC3 | http://www.atstake.com |
| LinuxSniffer | http://www.hackpalace.com/usa/ |

| | |
|---|---|
| John the Ripper | http://www.openwall.com/john/ |
| wipe | http://www.phreak.org/archives/exploits/unix/log-tools/ |
| zap2 | http://www.phreak.org/archives/exploits/unix/log-tools/ |
| utmp.c | http://www.phreak.org/archives/exploits/unix/log-tools/ |
| wtmped | http://www.phreak.org/archives/exploits/unix/log-tools/ |

## References

[1] The Honeynet Project, *Know Your Enemy,* Addison-Wesley, Boston, 2002.

[2] E. Skoudis, *Counter Hack*, Prentice Hall, Upper Saddle River, N.J., 2002.

[3] B. Hatch, G. Kurtz, J. Lee, *Hacking Linux Exposed: Linux Security Secrets & Solutions*, Osborne/McGraw-Hill, New York, 2001.

[4] Aleph One, "Smashing the Stack for Fun and Profit,"*Phrack*, vol. 7, no. 49, Nov. 1996.

[5] http://rr.sans.org/authentic/l0phtcrack30.php

[6] http://www.atstake.com/research/lc3/

[7] daemon9, "IP -spoofing Demystified," *Phrack*, vol. 7, no. 48, June 1996.