GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials (Security 401)"
at http://www.giac.org/registration/gsec

The Quest for Root – Hacker Techniques and UNIX Security
Michael A. Scott
September, 2002
GSEC Practical v.1.4b

ABSTRACT

The holy grail to the UNIX hacker is to obtain superuser or `root` account status. In this achievement, there is the likelihood of system compromise with complete control over processes, users and file structure. The ever-increasing implementation of UNIX servers connected to the Internet will inevitably lead to increased security compromises. The general techniques used to gain and maintain `root` access including use of trust relationships, abuse of setuserid programs, trojan horses and rootkits are discussed. An analysis of a newly discovered rootkit, Tuxkit, shows clear signatures that can be used for intrusion detection and we set out general strategies for security hardening our systems from these types of attack.

INTRODUCTION

A brief history of UNIX and security

The development history of UNIX operating systems (OS's) is complex due to the multitude of various versions and their hybridisation. A thorough treatment can be found in Salus [1]. Regardless of the exact system, all UNIX flavours whether commercial e.g., HP-UX, AIX, or the popular 'free' distributions e.g., Linux, FreeBSD etc all have roots tracing back to the two monumental efforts in the early 1970's by academics at Berkeley and Bell Labs in their development of the BSD and SYSV systems, respectively.

Back in 1965, Bell Labs worked on the mammoth and ambitious Multiplexed Information & Computing Service (MULTICS) system which is a modular OS designed to be continuously operable rather like a utility with inherent strong security features [2]. Largely due to defence related development, the MULTICS security model was necessarily restrictive having a high level of assurance provided by the Access Isolation Mechanism. This mechanism enforced classification of information based on multilayers of confidentiality and in 1985, some 15 years prior to it's demise, MULTICS was granted a B2 security rating by the US government [3]. Frustrated by the restrictions of the project, the Bell Labs developers wanted to create a new flexible operating system that had a freer security model with multi-user and file sharing capabilities aimed to support open software development. Hence, UNIX was born.

In contrast to the MULTICS multilayer restrictive security model, UNIX fundamentally has a 'binary' all or nothing approach - the 'superuser model'. The UNIX filesystem is the most basic tool for enforcing security and controls the nature of information storage and access privileges. Administrative tasks are performed by the superuser or `root` whilst all other 'normal' users have restricted access and should be unable to access

© SANS Institute 2000 - 2002, Author retains full rights.

critical files and system processes. This conforms well to the principle of least privilege that users should have minimum privileges for the tasks they need to perform.

In the early days, UNIX was the operating system of choice for the academic, research and computing industries where there was only a small number of interconnected machines, for example, in a laboratory environment. Security was not a great concern and a sense of complacency developed in UNIX circles in these small 'trusted' networks. In contrast, nowadays we are routinely connected to large, complex networks of which the Internet is a superset and complacency regarding network security is no longer tolerable. UNIX users had wake-up calls to network security through two very different but equally famous attacks, the 'Morris Internet Worm' (1988) and the 'Mitnick Attack' (1994).

In contrast to Windows and Macintosh systems, viruses haven't had a large presence on the UNIX circuit. Viruses need both user interaction and 'host' programs to activate them and the well-defined privilege and file structure in UNIX protects the system against substantial virus propagation and damage.

A more significant threat to UNIX security are self-replicating worms that when released in the wild automatically propagate full working versions of themselves across local and networked machines without user interaction. Unlike contemporary and hostile worms like Code Red, the Internet worm was designed to be relatively benign. The details and consequences of the Internet worm are well documented elsewhere, however, in summary it's successfully replication exploited vulnerabilities in `fingerd` and in the DEBUG method of `sendmail` [4]. In essence it was an experiment by Morris to measure replication across the Internet. The worm did not actually alter files or cause direct system damage but a bug in the code increased the intended replication rate and some 6,000 VAX and SunOS machines were infected across the USA. At this time, that was a massive scale availability attack on hosts within the fledgling Internet.

The famous Mitnick attack was the antithesis of the Morris worm. Instead of targeting many systems on the Internet, infamous hacker Kevin Mitnick launched a well-planned series of attacks against on the systems of Tsutomu Shimomura, an accomplished security professional [5]. We will outline the details of the attack later in our discussion of UNIX `root` account attacks.

More recently, in early 2001, we saw a boom time for UNIX exploits with the discovery of the Adore, Lion and Ramen worms [6,7,8].

The increased rate of UNIX attacks with time is due to the fact that UNIX systems are often set-up as servers (e.g., file and webservers) and are connected to the Internet. In the default install setup, there are normally a myriad of services running that, from a security standpoint, need to be reviewed carefully. Since the probability of successful attacks increases with the number of open ports and number of discovered vulnerabilities, we will continue experience a continued rate of increase in the number of systems affected.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The open-source UNIX variants

In recent years, there has been an explosion of interest in 'open-source' variants of UNIX such as Linux and OpenBSD. The philosophy of the open-source software initiative is the free distribution of application and source code. By free we mean both liberated and without charges.

At present, we are seeing an exciting and promising rapid adoption of Intel-powered Linux systems by home users and hard core businesses. Aside from the obvious financial benefits of open-source UNIX, the impetus is that users benefit from the flexibility and stability of the OS whilst harnessing the powerful processing of the Intel architecture. Indeed, HP and Sun are giving a determined market push towards Intel-based Linux servers aimed at large enterprises who want to migrate from conventional UNIX mainframes. Early adopters include the large seismic services company Western Geco and Morgan Stanley [9,10]. Other promising boosts for UNIX are the continued successful implementation and development of the Apple Inc.'s OSX which is derived from Berkeley's BSD UNIX and the release onto the Personal Desktop Assistant (PDA) market of Sharp's Zaurus wireless Linux system [11,12].

From a security viewpoint, the advantages of open-source software development are the opportunity for the community to patch potential security vulnerabilities or to extend the original capabilities of the software. On the other hand, this introduces a 'double-edged sword' scenario with potential for exploitation of unpatched security holes and embedding of malicious code into legitimate applications by less benevolent programmers.

The root user and hacking UNIX

An analogy can be made between implementing network security and a carefully planned military strategy. It is crucial that we understand the enemy, the threats and methods used and to learn from the lessons of history. The latter is very important as the popular open-source distributions have shared ancestry with the older more established 'flavours' of UNIX. As a result, a vulnerability found to affecting one system may also affect others and many previously tested and solved security issues have returned to haunt the 'new' systems. This 'old wine in new bottles' scenario is likely to present ever-increasing security issues. Indeed, recently, Bruce Schneier has postulated that the biggest threat to network security into 2003 will be the "ever-increasing tsunami of old attacks that continue to do the damage" [13].

It is dangerous to have preconceived ideas about the hacker's skill, background and intentions but, generally speaking there are two main hacker genres in the Black Hat community. There is the proficient 'professional' hacker who carefully footprints and targets specific companies or groups of companies and uses a judicious combination of tools and stealth to execute his/her attack. On the other hand, there is the 'script kiddie' who is indiscriminate and trawls the Internet searching for victim servers having specific vulnerabilities to which he/she can apply exploit code. The more skilled script kiddie is

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

capable of analysing and writing their own source code though the majority use readily downloadable and pre-packaged exploits to perform attacks, sometimes rather crudely and 'noisily'. As security professionals, we can use these attack signatures to detect exploits in the wild and help avoid further compromises.

Regardless of the hacker genre, the common motivations include some or all of the following:

(1) Use of the victim's network bandwidth and/or processor time e.g., download software and run unauthorised processes
(2) Pilfering and the corruption of data e.g., web page defacement, stealing of 'corporate jewels'
(3) Storage of illegal files e.g., media files, warez
(4) To gain a platform for attacks on other systems e.g., distributed denial of service (DoS) attacks

The framework of an idealised hacker attack on server `victim` is depicted in Figure 1. Firstly a series of reconnaissance probes is run against the target and potential vulnerabilities sought. Once a suitably vulnerable system is found the hacker executes known exploits against the vulnerable service and may gain administrator level access to the system. Having achieved this, the hacker has complete control of the system. The choice is then to either keep a low profile and install packet sniffers to monitor network traffic for interesting events or to launch a full scale information warfare attack. By information warfare we mean the offensive corruption or exploitation of information systems in order to gain advantage over 'the enemy'.

In UNIX systems, to achieve any of the above goals, the 'Holy Grail' to the hacker is to obtain `root` user status either directly or by a multistep process. By obtaining `root`, he/she becomes the superuser with UserID (UID) of 0 and has the highest level of privileges on the entire system. Such privileges include: unconditional access to system resources; user account administration; ability to `kill` and `renice` processes; file manipulation and device control [14]. Consequently, it is of utmost importance that the `root` account be secured as tightly as possible.

From the attacker's perspective, obtaining and maintaining `root` access is tricky, requiring skill and stealth. Indeed, it is essential throughout the entire attack lifecycle that their presence on the victim's server goes unnoticed and that they use suitable methods to ensure a covert return to the compromised machine.

The aim of this paper is to assess the security issues that surround gaining `root` access and also how the attacker can covertly hide and maintain his presence. Finally in the light of these methods we describe ways that we can help to secure our UNIX systems from attack.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

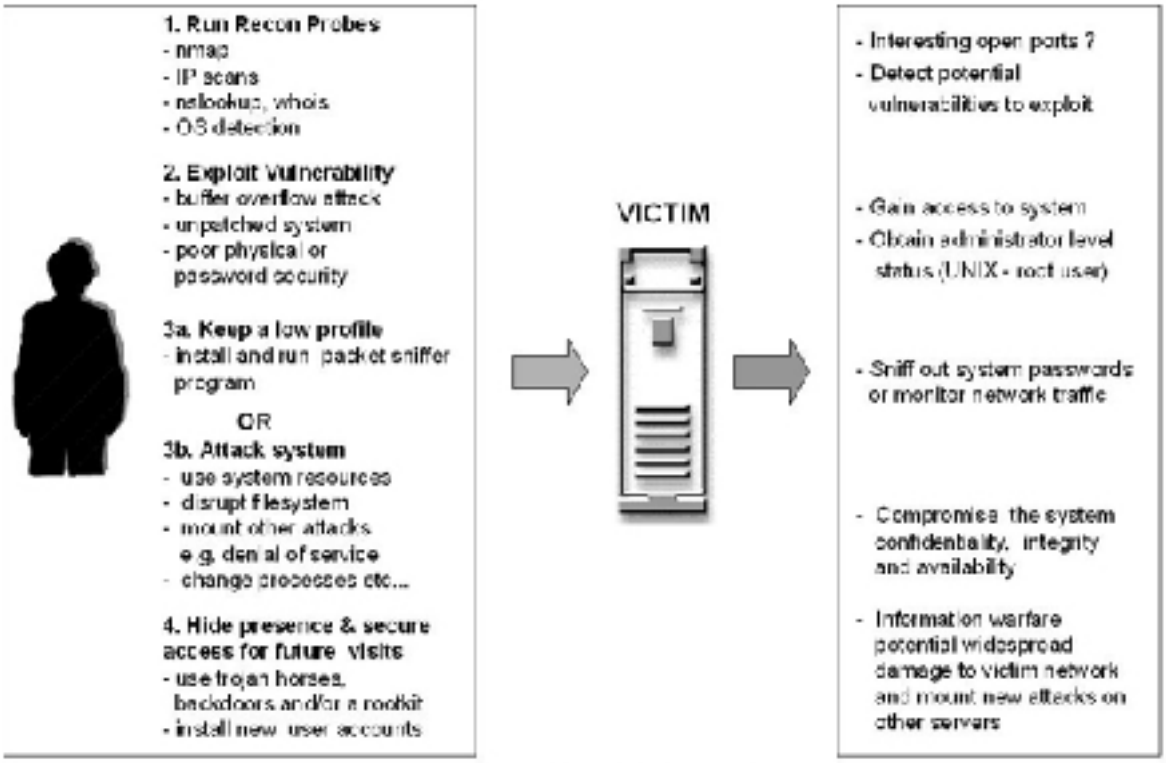


Figure 1 - **Framework of an idealised hacker attack**

METHOD

A test local area network was setup to perform security assessments and consisted of two PC's connected to a 10/100Mbps Linksys Workgroup hub (Model EFAH05W). Although the studies concentrate on the Linux system, the concepts described herein are relevant to other UNIX based systems. The main system `goodguy` was running Red Hat Linux 7.3 and was secured hardened whilst `victim` was used for security assessments and was running Red Hat Linux 7.2.
Throughout the testing process, `victim` was isolated from the wide area network and some important services were left running from the default install. These included `telnet`, `finger`, `ssh` and `rlogin`. In cases where original source code was used, the GNU C compiler `gcc` was used to produce binary executable files.

ABUSING ROOT

1. Exploitation of trust relationships

System administrators frequently have to work remotely between servers and for this purpose, UNIX provides us with a handy set of tools: the r-commands that were developed in Berkeley's BSD UNIX and include remote shell (`rsh`), remote executable (`rexec`) and remote login (`rlogin`). The advantage is that systems can be configured so that the `root` user on a remote host running a service at a privileged port (port

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

below 1024) is trusted and can connect at any time without re-authentication. In practice this is achieved by adding trusted hosts IP or names to the `.rhosts` file or `/etc/hosts.equiv`.

History provides us the best examples of security problems associated with the r-commands and both the Morris Internet worm and Mitnick's attack exploited the trust relationship concept to breach system security. The Internet worm checked for trusted hosts in the above files on each system it infected and, if possible, tried to use the `rexec` and/or `rsh` commands to infect further machines [4].

The Mitnick attack was a full scale Confidentiality, Integrity and Availability (CIA) attack on Shimomura's systems and data. Excellent descriptions of the intrusion are given in Northcutt and the extract of Shimomura's original `tcpdump` posting on comp.security.misc [15,5]. The attack was founded on the discovery of a trust relationship between two of Shimomura's Solaris systems , `server` and `x-server`. A summary of the main sequence of the attack events and is the best example possible to discuss the exploitation of trust relationships to gain `root` access.

(i) Reconnaissance
In accordance with Fig. 1, Mitnick performed substantial reconnaissance on the victim's systems from the `root` account on external host toad.com using `finger`, `showmount` and `rpcinfo` commands. He established that the `login` port was vulnerable and that there was a trust relationship between the two systems. Armed with this knowledge, he knew that if he could pretend to be `x-server` then he could then login to `server` without re-authenticating as explained above.

(ii) IP Spoofing
A DoS attack was used to silence the now trusted host `server`. Mitnick spoofed the address of a random, deliberately unused IP address (130.92.6.97) to initiate a SYN flood attack on port 513 (`login`) of `server`. The repeated bombardment of TCP SYN packets produced a rapid series of half-open connections since the normal TCP three-way handshake could not be completed [16]. The SYN-ACK replies could not be acknowledged by the offline system with the spoofed IP and the `server` connection queue table was filled using eight TCP packets producing a denial of this service and effectively gagging `server`.

(iii) Sequence Number Prediction
Having silenced `server`, any SYN-ACK packets sent from `x-server` would still return to the silenced trusted host IP and Mitnick was 'flying blind', i.e., he could not see these replies. In order to maintain his connection, he had to predict the initial sequence number (ISN) behaviour of `x-server` and craft correct ACK packet replies. Cleverly, to achieve this, Mitnick used another connection from apollo.it.luc.edu to send repeated SYN's with incrementing ISN's to find that there was a predictable difference of Δ=128,000 between each ISN of the of the `x-terminal` shell SYN-ACK packets.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

(iv) Session hijacking using root
By successfully masquerading as the trusted host he sent a remote shell packet equating to the command:

```
server # rsh x-terminal "echo + + >> /.rhosts"
```

which allows trusted `root` user permissions to all users and hosts without re-authentication. He then reset all the half-open connections caused by stage (ii) by sending RST packets from the spoofed IP address.

Having successfully gained access to the `root` account, Mitnick stole and transferred some files to the well.com domain and Shimomura's subsequent discovery of the attack helped lend to Mitnick's arrest [17].

2. Setuserid (SUID) processes and shells

Setuserid (SUID) is an extremely useful and unique feature of the UNIX system that enables normal users to elevate their privileges and execute commands that they would not normally be allowed. This transiently relaxes the constraints of the superuser security model somewhat and adheres to the principle of least privilege. This ensures that the user should not be logged into the `root` account simply because they need to run a couple of commands with privilege.

A SUID process is indicated by '**s**' at the file permission owner execute bit. When the process is executed, the effective userid (UID) of the program becomes that of the owner of the file. The well known `passwd` command is an example of a SUID command:

```
$ l –l /usr/bin/passwd
-r-s--x--x     1 root  root 15104 Mar 14 01:44 /usr/bin/passwd
```

we see that the SUID bit is set and the owner of the process is `root`. Any user changing their password transiently runs that process with `root` privilege. Most SUID programs are owned by `root` and we will refer to hereafter as 'SUID `root`'. SUID presents one of the biggest security threats in the UNIX OS and it is very important that we monitor the files are carefully.

Shells can also be made SUID root and normally shells that are owned by `root` have file permissions rwxr-xr-x. However, we can easily make SUID `root` copies providing we can access a `root` account in the first place.
The following example shows how to make a copy of the Bourne Again Shell (`bash`) from `root` on Linux and place it deep within the file structure of the system. The inconspicuous header filename `Xms.h` is given to the shell to mask it's true nature:

```
#    cp /bin/bash /usr/X11R6/include/X11/extensions/Xms.h
```

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Then the file access mode is changed by setting the SUID bit

```
#     chmod 4755 /usr/X11R6/include/X11/extensions/Xms.h
```

Hence we have an innocent looking SUID `root` shell that can be run by the normal user and used for returning attacks. Hackers often hide files deep within 'busy' directories including `/dev` in the hope that the security or system administrator misses the rogue file amongst the legitimate system files.

The same technique can be applied to `root`-owned files. We can also make SUID `root` copy of the `vi` editor called `mal-editor` using the same methods above. This file runs as `root` and allows to reading of root privileged files for reading. For example, the following command run as normal user lab

```
[lab@goodguy /]$ /tmp/mal-editor /etc/shadow
"/etc/shadow" [readonly]…
root : $1$73EnHydfsdLitE…:11841
```

allows one to view the encrypted shadow password file that would have been previously 'permission denied' to the normal user.

Clearly whether a program or a shell, the ability to make SUID `root` copies is attractive to hackers as it provides a mechanism for gaining future `root` privileges from a normal user shell without resorting to drastic measures like adding extra user accounts to `/etc/passwd`.

Since creating SUID root executables requires `root` access in the first place, the hacker could use a variety of diverse methods to get `root`. Examples of tried and tested techniques include taking advantage of a careless system administrator who leaves an open `root` shell while taking a break, using social engineering to coax account passwords or by using exploits across a network to spawn a `root` shell.

In common with other UNIX systems, the version of Linux used in the present study allows only those system binaries that are listed in the file `/etc/shells` to run as shells. Nevertheless, the new shell created in the above steps can be used to run *scripts* as `root` and potentially cause grave damage to the system.

3. Trojan horses, backdoors and rootkits

Since the earliest days of system hacking, attackers have tried to develop methods termed backdoors that allow them access back into previously compromised systems. The impetus is that having already done the hard work in exploiting a security hole to get `root`, the attacker needs a easy method of getting back into a machine even if the

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

sysadmin changes account passwords. Ideally the backdoor should be a quick and covert method that avoids being logged. Enter trojan horses and rootkits.

The methods described in section 2 to maintain `root` access are rather unsubtle, though commonly used by inexperienced hackers and those that are already inside the network. It would take an inexperienced or unwary system administrator not to notice the presence of the new files on the system. A much more elaborate and cunning method of maintaining access and fooling the sysadmin is to use trojan horses.

A trojan horse is a class of malware that on the outside appears to be a normal valid application but when executed does something completely different to the original intention whilst appearing normal to the user. Many different trojans exist, perhaps the most famous being the client-server remote control trojans SubSeven and NetBus affecting Windows [18,19]. In these trojan programs, system security is circumvented in the background and the victim has been previously infected by the server version of the trojan code allowing remote network connections form the client and remote control of his system.

In UNIX the biggest trojan threat are those that replace legitimate system binary files but covertly sabotage system security and are termed 'trojaned binaries'. Trojaned binaries are most commonly packaged within a rootkit which is a suite of tools used by the hacker once he gains `root` access to ensure future system entry with minimal logging [20]. Typically, rootkits come as tarballed source code that the hacker would download in the background using the freely available GNU `wget` - a persistent web browser. These files would then be compiled locally as `root` using the GNU compiler.

During the course of this research study, the author discovered a relatively new Linux rootkit has been observed globally in the wild by perplexed sysadmins. The rootkit called 'Tuxkit' has not been substantially investigated and was released by the Tuxtendo group in the Netherlands in December 2001 [21]. In this following analysis we look at the characteristics of this rootkit and in order to help aid intrusion detection and removal.

3.1 Analysis of the Tuxkit Linux rootkit

A simulated rootkit attack was replicated on my `victim` server. Firstly the tarballed version 1.0 rootkit, `tuxkit-1.0.tgz`, was downloaded from the Internet and unpacked. It was clear from the README file that this rootkit was designed for the fast and easy kill and aimed at less proficient hackers. The unpacked tarball yields further zipped files tools.tgz, bin.tgz, cfg.tgz, lib.tgz, ssh.tgz which contain a series of precompiled binaries and not source code.

The total tarball filesize is 2.6MB of which 60% of which is taken up by utilities. These include `dos/virii.c` a denial of service script which performs a DoS attack on address <IP> within a time delay of <x> seconds. Additionally, to facilitate password and network traffic monitoring `ADMsniff`, a packet sniffing program is included which puts the specified ethernet device into promiscuous mode to monitor ethernet traffic.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The packet sniffer was executed and `ifconfig` shows correctly that the PROMISC flag is set and packet capture is possible.

It appears the main purpose of the rootkit is to facilitate access from multiple hosts on the Internet to the victim system. This is evidenced by the array of Internet Relay Chat (IRC) tools found in this rootkit (BitchX, Mirkforce and psyBNC) for remote connection and to potentially to launch attacks on other systems from the victim server. BitchX is a heavily modified IRC client, Mirkforce creates IRC virtual hosts from a server using unused IP addresses in the subnet whilst the psyBNC bouncer program allows a persistent IRC connection using a vanity host [22,23,24]. In addition, a precompiled program called `suidsh` produces a SUID root shell so that the hacker does not need to create on their own by methods outlined in section 2.

Prior to executing the rootkit, the trojan binaries were unpacked and the file attributes (filesize and datestamps) were compared with the real binaries as shown in Table 1 of Appendix A. In general, we can see that the trojans are smaller and have different timestamps than the real binaries as one would expect. Message digest (md5) checksums were then recorded of the critical binaries on my system using md5sum command and are shown in Table 2 of Appendix A.

The Tuxkit rootkit was then executed as follows:

 # ./tuxkit <passwd> <port1> <port2>

where the user-defined values were passwd = 'hack3r', port1 = 6789, and port2 = 6969 and the resultant screendump show in Fig. 2.

The slogan "We hope to please you kiddies!" confirms the intended audience for the rootkit. After creating the `/dev/tux` directory and trojanising critical system binaries including `crontab`, `ifconfig`, `netstat`, `ps`, `syslogd and sshd`, psyBNC is set to automatically listen on port 6969. The syslog daemon is killed which masks the presence of Tuxkit from the log files. We should note that unlike the t0rnkit, this rootkit runs under user-defined ports not defaults and is more tricky to predict [25].

The author checked the `root` mail account as routine and noted there were repeated attempts to mail two recipients with the IP address of the victim server, backdoor SSH login password and listening ports.

The installation of the trojaned secure shell `ssh` daemon is used as a backdoor. By executing `ssh` at port1 and providing the backdoor password `root` access , a root shell can be obtained. The main goal of the trojaning is to mask the existence of the rootkit and the backdoor `ssh` in `netstat`, file system commands and logs. The startup scripts are modified to ensure that the `ssh` backdoor is run on boot.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Analysing the /lib directory shows that system process monitoring is subverted through trojaning of the `libproc.so.2.0.7` library file. This file is a symbolic link to `libproc.so` file and is part of the `procps` RPM that monitors important system processes. These include `ps`, `free`, `vmstats`, `skill` and `uptime` [26]. We test the trojaning by running `ps` command:

```
PID TTY          TIME CMD
17984 pts/2    00:00:00 su
17988 pts/2    00:00:00 bash
18022 pts/2    00:00:00 ps
```

and we have a clearly very sanitised picture of the reality.
However, running `ps` with `-ef` options produces a more detailed listing

```
UID        PID  PPID  C STIME TTY          TIME CMD

root       783     1  0 Aug24 ?        00:00:02 /usr/sbin/sshd
:
root       847     1  0 Aug24 ?        00:00:00 crond
xfs        899     1  0 Aug24 ?        00:00:00 xfs -droppriv -
lab       2157  2155  0 Aug24 ?        00:00:00 nautilus --sm-
root     13590 13586  0 Aug30 pts/1    00:00:01 bash
:
:
root     13717 13590  0 Aug30 pts/1    00:00:00 ./ADMsniff eth0
```

and we clearly see the presence of the `ADMsniff` packet sniffer and the trojaned ssh running.

Probing active network connections using `netstat -an` shows nothing out of the ordinary and has been trojaned well. Instead list open files, `lsof`, was run with pattern matching for listening ports, `lsof | grep LISTEN`

```
xsf         274 root    3u  IPv4        565                 TCP
*:6789 (LISTEN)
portmap     586 root    4u  IPv4        896                 TCP
*:sunrpc (LISTEN)
:
:
xinetd      816 root    8u  IPv4       1173                 TCP
*:telnet (LISTEN)
X           957 root    1u  IPv4       1326                 TCP
*:x11 (LISTEN)
psybnc     1678 root    3u  IPv4       7577                 TCP
*:6969 (LISTEN)
```

© SANS Institute 2000 - 2002, Author retains full rights.

```
xsf       16549 root    3u  IPv4     508237                TCP
*:6688 (LISTEN)
```

The listing clearly shows that `psybnc` and the rogue `ssh` are in listening states for connections and that the rootkit authors did not trojan the important tool `lsof`. Editing the `crontab` file show that `psybncchk` is set to run every minute, daily. The trojaned `ifconfig` shows

```
eth0      Link encap:Ethernet  HWaddr 00:50:8B:92:44:00
inet addr:192.168.0.51  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

and the promiscuous mode of the ethernet interface is hidden despite the fact we know it is active after running the packet sniffer `ADMsniff`.

Running `ls` hides the presence of the rootkit and associated files very effectively. It was noted that the rootkit removes the original tarball and that the main signature of the Tuxit is the presence of the `/dev/tux` directory that is hardcoded into the binaries.

Apart from the active network signatures of the rootkit, the main tool on the side of the security professional is file integrity analysis. After installation, the file sizes and time/date stamps of the trojaned binaries are exactly the same as the originals and could fool sysadmins. This is presumably the function of the extra program, `sz`, found in the `/bin` directory. When executed, `sz` asks for two filenames and presumably equates the file sizes possibly by zero-padding the difference, if necessary.. However, by running md5 checksums on the trojaned files and comparing with the original values in Appendix A Table 2, we can see that the discrepancies indicate that `md5sum` has not been trojaned. This provides us with a crucial way to detect the intrusion.
Finally, a covert connection to `victim` was initiated from goodguy

```
[root @ goodguy root]#  ssh 192.168.0.51 –l root –p 6789
```

with the password hack3r entered at the prompt. This gave the `root` prompt with complete remote access to `victim`.

It was observed that `chkrootkit` v0.36, the popular and useful rootkit checker, does not specifically detect this rootkit [27]. It recognises known rootkit signature files within some of the trojaned binaries and tags them as 'INFECTED' but not the trojaned `ls` or `login`. Moreover, it wrongly signals possible Loadable Kernel Module (LKM) rootkit infection. LKM rootkits are outside the scope of this study but more information can be found in ref. [28]. It is noteworthy that, on their website, the rootkit authors state the intention to release the next version of Tuxkit as an LKM rootkit .

A schematic summary of the processes run under Tuxkit v1.0 are shown in Figure 3.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.



Figure 2: Screendump of the Tuxkit rootkit v1.0 running under Linux

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

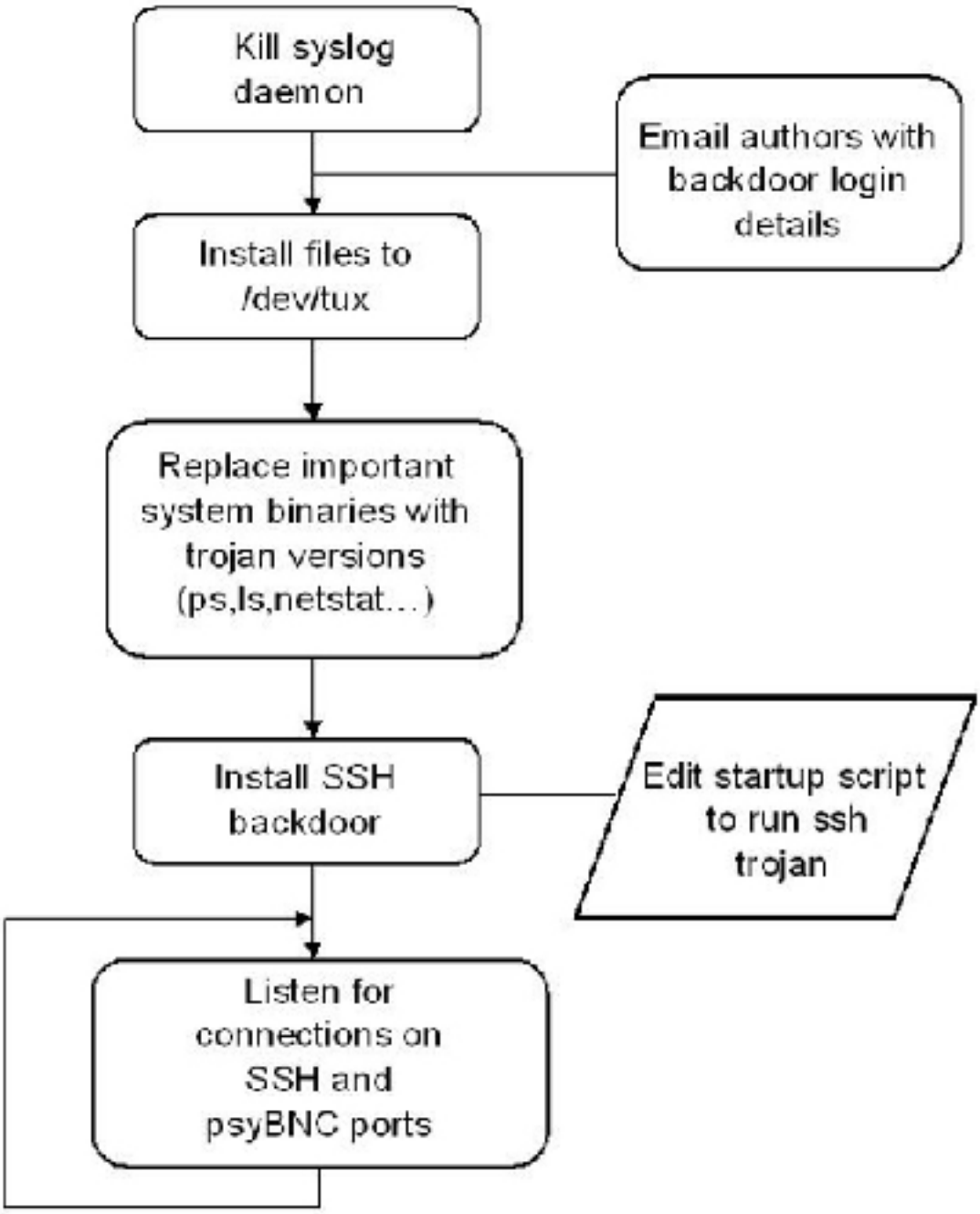


Figure 3 - Process chart of Tuxkit rootkit v1.0

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

DISCUSSION

In this paper we have described various methods in which attackers can use to gain and maintain access to the superuser or `root` account in UNIX systems. In particular, we described that by exploiting the weak IP-dependent authentication in the Berkeley r-commands and trust relationships we can gain `root` access at anytime without re-authentication. Exploiting trust in the way that the Internet and Mitnick attacks did is highly effective, albeit rather unsubtle by today's standards of security. Even now though, system administrators need remote access and often the convenience of the r-commands takes precedence over security. These classic attacks are a stark reminder to us about the risk of relying on trust relationships and clear text authentication. Instead, it is far better to use the secure shell (`ssh`) which uses encrypted network traffic , thereby improving security. Additional assurance is obtained by making sure that systems are up-to-date with patches especially critical vulnerabilities and kernel updates.

Since these early days, the size of networks has exploded and there has been greatly increased awareness for the need to maintain good network security practice. Any good security strategy would employ the use of perimeter defences including a filtering router or firewall to block incoming connections from risky services such as the 'r-commands'. A correctly configured firewall would have blocked access to spoofed IP address packets of the type sent by Mitnick.

To add to our general strategy for UNIX security we should routinely run the vulnerability probes on our own networks as this is exactly what the hacker does in the initial stages. Using the indispensable `nmap` vulnerability and port scanner, even in vanilla (TCP only) scan mode we can see what services are running and what ports they map to [29]. Moreover, combined with using the Security Administrator's Integrated Network Tool (SAINT) provide a good baseline for identifying potential vulnerabilities and patching them [30]. The golden rule is to turn off the services that we don't need.

We examined the commonly abused SetUserID (SUID) commands and shells. It is obviously very important to know exactly what SUID `root` programs exist on a given machine and therefore be able to demarcate new files. Using the command below will to check all files with the SUID bit set

```
# find / \(-perm –004000 \) >> suidfiles
```

and output to the ASCII file `suidfiles`. Appendix B shows the SUID `root` files on the test Linux system, `goodguy`, used in this study. Once the SUID programs have been evaluated it is a good idea to store them securely perhaps on a CD-ROM. The process is useful as it allows you to audit the privileges on a particular system and remove the SUID bit if the command does not need to run SUID and conform to the principle of least privilege. Garfinkel and Spafford give a very thorough review of SUID files as well as SetGroupID files (SGID) compensating for different vendors of

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

UNIX [14]. It is also good security practice to turn off SUID when mounting from foreign filesystems with the `nosuid` option e.g.

```
mount –o nosuid  machine:filesystem  /dir
```

File integrity checking of the OS is an absolute must and should be employed before the server is connected to the network and ideally after installation to ensure accurate data. This, as we have seen in the analysis of rootkits, provides an essential mechanism for detecting file changes once an attacker has compromised the system, particularly where trojaned binaries are used. A secure, remote log of the checksums should be kept or better still, the use of file integrity programs such as Tripwire to database the checksums and routinely check system file integrity should be used [31]. Tripwire, if baselined correctly provides an excellent means of auditing file system change.

To check specifically for rootkits, `chkrootkit` is a useful tool and is capable of detecting a variety of rootkits, trojaned binaries and whether the interface is running in promiscuous mode [27].

The author considers a host-based intrusion detection program to be another essential for our security arsenal and uses Portsentry v.2.0 for Linux by Psionic software on the security hardened system, `goodguy` [32]. This program monitors a specified set of TCP/UDP ports for portscans and Portsentry is configured to run when the ethernet interface is up and to e-mail `root` whenever an attack alert occurs. The remote IP address is added to TCP wrappers's `/etc/hosts.deny` so one must be careful to ignore internal network scans and also make sure that Domain Name Servers and trusted IP's are excluded. Otherwise there will be a denial of service.

The type of rootkit studied provides an convenience with pre-packaged trojans that do not even require to be compiled. Both Tuxkit and t0rnkit are showing that these dangerous types of convenience rootkits can cause a great deal of trouble. The increased number of less security aware individuals will make themselves targets for script kiddie's armed with these kits.

A combinational strategy to security is essential and using the tools at our disposal, knowing our enemies methods and implementing good network security designs will prevent our `root` accounts from being sabotaged by the badguys.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

APPENDIX A: File attributes of trojaned and normal binaries in Tuxkit rootkit

Filename	Size / bytes (normal)	Time & Date (normal)	Size / bytes (trojan)	Time & Date (trojan)
df	26812	August 9	27112	December 26
dmesg	4252	July 31	3640	December 26
login	17740	July 31	3980	December 26
ls	45948	August 9	42952	December 26
vdir	45948	August 9	42952	December 26
netstat	83132	July 31	58228	December 26
ifconfig	51164	July 31	36356	December 26
syslogd	26972	July 31	28324	December 26
crontab	21280	June 25	29052	December 26
dir	26972	August 15	42952	December 26
killall	12096	July 21	14400	December 26
locate (symlink)	7	August 23	9144	December 26
pstree	12284	July 21	14532	December 26
top	34924	August 28	37844	December 26
updatedb (symlink)	7	August 23	4394	December 26
find	47516	June 25	55220	December 26
tcpd	24844	June 25	18660	December 26
du	25788	August 9	25592	December 26
ps	63180	August 28	62748	December 26

Table 1 – File attributes of selected critical binary files and the trojan versions before rootkit installation (all date years are 2001)

Filename	md5 checksum (normal)	md5 checksum (trojaned)
df	f70b403e05ab12b4cec4b0c4c53228ce	1ed369095b0ecece319c47d66b6b66c1
dmesg	e7000edfb73b09b659555e9902650f7b	c67f34eeab989275618742caf3086dca
login	c877f8a0595513ee68d857e6e9754a3f	f2cc26e4c0ca7083c35780d8015a2961
ls	69ee580c4bd6afa63aed49076c535f62	83ea81b1b39e593d3387633f25104eae
vdir	65e9e2159a40569d79fd656d8c8311c9	2d76827d4a4bb465fc7ef18d4147bcc0
netstat	f7d2bf3b53ee2c9145afff5f2edecc45	32bb3d9fd18b50708f01c52a7b28a0e6
ifconfig	415980d501beaa167f33dab16d80a817	8cb5f402bfda848bcb684a2cd7e68bb8

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

syslogd	3b9a74413eaf20061d855270d64e78c4	0df53866813d70acce2cbbbe9d54e2d4
crontab	e94bdec4e91efab48a47294389a1c54d	8715009f40538ff7b734783c834457c8
dir	03d1235a03bc6ce8322bcfede328fa92	4e1e0354019075fe10b03c647dbe05a9
killall	952e3ae67b2b77d9073a7c66c4219cf1	9265f715e7d0a23c2e93a4f3d6fbc34a
locate	4de85260e08fd5e1f8e27a0139e591ff	3c656ef9803a7badda439321c4ffa784
pstree	a8608f8c824d05df1876a0ea3164c2f4	4f887e6728e76fcd7c9e1a2f86d37dcb
top	d71d83ad4c4af666e5fc49f270d88403	40f56e0752c1890f186db9c3f7533c6c
updatedb	4de85260e08fd5e1f8e27a0139e591ff	4ed4fffa4e904e9f3eb5de74ca234f0c
find	31d920f052841d5a250496036ea176c4	bb35340fc4bc49683c76f32e85bb325f
tcpd	9b31c04dbd430656f822253c7565b6c5	6fa1a757b57cb38363553d8b953b2d41
du	d295a486e04f96d928a8b15a9833b3a6	378a08f3f042a91fcf654b08922da003
ps	6d3abf4efc9235e4eb5dc540d61d42fa	f4dc73fa2c474acfbdd6273e8a82dfd9

Table 2 – md5 checksums of critical binary files and the trojan versions after executing the Tuxkit rootkit

APPENDIX B: List of SUID `root` programs on test Linux system `victim`

/usr/sbin/ping6	/usr/sbin/traceroute6
/usr/sbin/sendmail.sendmail	/usr/sbin/userhelper
/usr/sbin/usernetctl	/usr/sbin/userisdnctl
/usr/sbin/traceroute	/usr/sbin/suexec
/usr/X11R6/bin/XFree86	/bin/ping
/bin/mount	/bin/umount
/bin/su	/sbin/pwdb_chkpwd
/sbin/unix_chkpwd	/usr/bin/chage
/usr/bin/gpasswd	/usr/bin/at
/usr/bin/passwd	/usr/bin/chfn
/usr/bin/chsh	/usr/bin/newgrp
/usr/bin/crontab	/usr/bin/lppasswd
/usr/bin/ssh	/usr/bin/rcp
/usr/bin/rlogin	/usr/bin/rsh

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

/usr/bin/sudo	/usr/lib/mc/bin/cons.saver
/usr/sbin/ping6	/usr/sbin/traceroute6
/usr/sbin/sendmail.sendmail	/usr/sbin/userhelper
/usr/sbin/usernetctl	/usr/sbin/userisdnctl
/usr/sbin/traceroute	/usr/sbin/suexec
/usr/X11R6/bin/XFree86	/bin/ping
/bin/mount	/bin/umount
/bin/su	/sbin/pwdb_chkpwd
/sbin/unix_chkpwd	

REFERENCES:

[1] Salus, Peter H. A Quarter Century of Unix. Addison Wesley, 1994.

[2] Van Vleck, Tom (Editor). "Multics – General Info and FAQ" , 31 Oct 2000
URL: http://www.multicians.org

[3] National Computer Security Center
TCSEC Criteria Concepts Trusted Computer System Evaluation Criteria (TCSEC)
URL: http://www.radium.ncsc.mil/tpep/process/faq-sect4.html#Q18

[4] Page, Bob. "A Report on the Internet Worm" , November 7 1998
URL: ftp://coast.cs.purdue.edu/pub/doc/morris_worm/worm.paper

[5] Gulker Chris. "Technical Details of the Attack", September 17 2001
URL: http://gulker.ra.com/ra/hack/tsattack.html

[6] SANS GIAC Global Incident Analysis Center. "Adore Worm
Version 0.8", April 12, 2001
URL: http://www.sans.org/y2k/adore.htm

[7] SANS GIAC Global Incident Analysis Center. "Ramen Worm
Version 0.4", April 12, 2001
URL: http://www.sans.org/y2k/adore.htm

[8] Vision, Max. "Lion Internet Worm Analysis". 2001
URL: http://www.whitehats.com/library/worms/lion/

[9] "Businesses turn to open-source system as vendors add offerings"
URL: http://www.informationweek.com/story/IWK20020207S0018

[10] Linux Journal. "Reuters Supports Adoption of Linux in Financial Services Industry". May 17 2002
URL: http://pr.linuxjournal.com/print.php?sid=104

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

[11] Apple Computer, Inc. “Mac OSX and the power of UNIX”. 2002
URL: http://developer.apple.com/unix/

[12] Lehrbaum, Rick. “Sharp's Zaurus SL-5500 Linux PDA”. May 13, 2002
URL: http://www.linuxdevices.com/articles/AT2134869242.html

[13] Schnieier, Bruce. “Know your Enemy - Threat: The Ever-increasing Danger of Old Attacks.” Computer Weekly 23 May (2002): 42-44.

[14] Garfinkel, Simson and Spafford, Gene. Practical UNIX & Internet Security. Second Edition, O’Reilly, 1996.

[15] Northcutt, Stephen. Network Intrusion Detection: An Analyst's Handbook . New Riders ,1999. 1-16.

[16] Naugle, Matthew G. “The Three-Way Handshake” Illustrated TCP/IP, Wiley Computer Publishing, John Wiley & Sons, Inc. 1998. Ch. 196

[17] Gulker Chris. “The Kevin Mitnick/Tsutomu Shimomura affair”, September 17 2001
URL: http://gulker.ra.com/ra/hack/index.html

[18] Crapanzo, Jamie. “Deconstructing SubSeven, the Trojan Horse of Choice”
URL: http://www.sans.org/infosecFAQ/malicious/subseven.htm, January 8, 2001

[19] PCHelp. “NetBus – BO’s Older Cousin.”
URL: http://www.nwinternet.com/~pchelp/nb/netbus.htm, November 25 1998

[20] Dittrich, Dave.”Root Kits and hiding/files/directories/processes after a break-in”.
URL: http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq, January 5 2002

[21] Tuxtendo group – source of the Tuxkit rootkit
URL: http://www.tuxtendo.nl/index.html
rootkit Tuxkit v1.0 was downloaded for security assessment purposes at
URL: http://archive.tuxtendo.nl/rootkit/

[22] BitchX IRC client, 2000
URL: http://www.bitchx.com

[23] ”Mirkforce / Hack reporting site”
URL: http://hackreport.magicnet.org/mirkforce-info.html

[24] Jestrix, “Introduction to psyBNC “, 2002
URL: http://www.netknowledgebase.com/tutorials/psybnc.html

© SANS Institute 2000 - 2002 Author retains full rights.

[25] Miller, Toby. “Analysis of the t0rnkit rootkit”
URL: http://www.sans.org/y2k/t0rn.htm

[26] Linux From Scratch: Version 3.3, “Installing Procps-2.0.7”
URL: http://www.tldp.org/LDP/lfs/LFS/chapter06/procps.html

[27] chkrootkit, “Locally checks for signs of a rootkit”
URL: http://www.chkrootkit.org/

[28] Zovi, Dino Dai. “Kernel Rootkits”, July 4 2001.
URL: http://rr.sans.org/threats/rootkits.php

[29] “NMAP - Free Stealth Port Scanner For Network Exploration & Security Audits”
URL: http://www.insecure.org/nmap/

[30] “SAINT scanning engine”
URL: http://www.wwdsi.com/products/saint_engine.html

[31] Stancin, Alexander. “Installing and Running Tripwire”, October 5 2001
URL: http://www.vrlteam.org/home.asp?vrl=library&adv=78

[32] Psionic Technologies, PortSentry v2.0
URL: http://www.psionic.com

© SANS Institute 2000 - 2002, Author retains full rights.