Securing Unix

CSC 348·648

WAKE FOREST
UNIVERSITY
Department of Computer Science

Spring 2013

Operating Systems and Security

- OS security controls consists of the following items
  1. Login and user accounts
     - Info about users are stored in *accounts*
     - This includes *privileges* granted to a user
     - *Identification* and *authentication* identify a user
  2. Access control
     - *Permissions* on resources can be set by the administrator
     - Permissions will depend on the user's identity
  3. Audits
     - Security involves *prevention* and *detection*
     - Mechanisms are needed to determine security violations

4. Configuration
   - Updating OS based on security needs

- We will focus on Unix security
  - It does not have a great reputation for security
  - Offers a set of features that can be used to enhance security

Unix Security Background

- Originally intended for one machine with multiple users
  - Main objective of security was to isolate mistakes
- Continually extended without regards to security
  - For example the Berkeley extensions (e.g. `rlogin`, `rsh`)
  - Support a LAN of machines under centralized control
    How are trust associations made?
- Internet RFC and ARAPNET extended support to WAN
  - Programs such as `telnet`, `ftp`, and `DNS`
  - However security was not the primary objective
- Sun contributions include NFS, NIS, and RPC
  - Sharing of computing resources *What about security?*

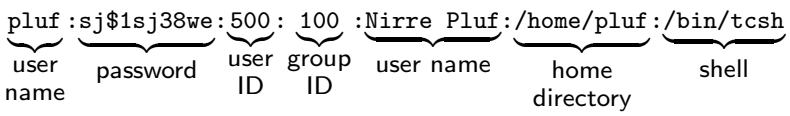
Basic Mechanisms

- Unix programs are run as *kernel* or *user*
  - Kernel has complete control of **all** resources
  - User has limited access to resources
- Access control is done using ACL

  Is this RBAC? Is `root` *an account?*

Login and User Accounts

- In Unix, users are identified by *user names*
  - Authenticated by *passwords*
  - Passwords are *encrypted* and stored in `/etc/passwd`
- `/etc/passwd` stores the following per user



  - All entries are : delineated
  - *userID* and *groupID* will be explained later
  - *IDstring* is the user's full name
  - *homeDirectory* is the home directory location
  - *loginShell* is the shell environment

- An example `passwd` file

```
root:nbBBu$1as$5kdjf:0:0:root:/root:/bin/tcsh
pluf:asdjj23jDJds:500:10:Nirre Pluf:/home/pluf:/bin/tcsh
nomed:$1skhQ87FH:510:10:Nomed Nocaed:/home/nomed:/bin/bash
```

  - If the password is empty, password is not required
  - If the password starts with an `*`, the user cannot login
    Why might this be useful to an admin?

- `passwd` is readable by any user, but only *root* can write to this file
  Any security risks associated with reading?

  How can a user change their password if the file is read only?

Shadow Files

- There is a security risk associated with a readable `passwd`
  - Crack programs are available determine passwords
  - Capture the password file then run a crack program
    (*off-line, this takes some time...*)
  - A *shadow password file* is used to address this
- If a shadow password file is used
  - An `x` is placed for the password in `/etc/passwd`
  - The actual encrypted password is located in `/etc/shadow`
  - The `shadow` file is readable only by `root`
- So if you want to crack passwords...
  - Obtain `passwd` and `shadow` then merge
    Requires `root` access, so why bother?

Encrypted Password Format

- Traditionally passwords were encrypted using DES
  - The function `crypt()` performed this task
  - However it supports 8 characters and 2 bytes of salt...
- Newer systems can use a different encryption methods
  - For example, SHA-256/512, MD5, ...
  - This is defined in `/etc/login.defs`
- The length of salt can vary as well in newer systems (*verify*)
  - Given encrypted value, number between $ and $ is salt length

```
pluf:$6$/2k7gC7o$pW4eCQ.MmB1AQ3jNiRrEpLuFrUlZeSJ1:14629:0:99999:7:::
```

Given n entries in the file, which should you crack first?

User Types

- `root`
  - Superuser, has complete control of the system
- Normal user
  - Users that can log in the system, have restricted access
- System user
  - System users cannot log in
  - Accounts used for specific system purposes
  - `nobody` (`http` requests) and `lp` (printing requests)

Unix User IDs

- Unix represents users by a *user name* and a *user ID* (UID) number
  - UIDs are linked to user names in the `passwd` file
  - Unix does not distinguish between users having the same UID
- Some UIDs have special meanings

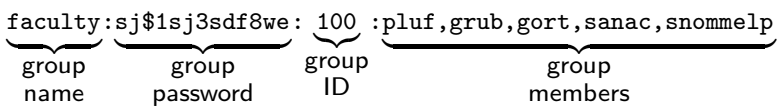
0	root
1	daemon
2	uucp
3	games

⋮

- N.B. any account with UID of zero is *root*, other common names include `toor` and `super`

Unix Group IDs

- Users belong to one or more groups
  - Collecting users in groups is convenient for access control
  - For example *guests* could be placed in one group
- Every user belong to a primary group
  - Similar to users, every group has an ID (GID)
  - The group info is stored in `/etc/group`



the **Root** *of all evil*

- Every Unix system has a user with special privileges
  - This *superuser* has UID of 0
- The `root` account is used for essential administration tasks
  - Adding users, auditing logs, etc...
  - As a result almost all the security checks are off
- Superuser can do almost everything
  - Become any other user (`su` *userName*)
  - Change `passwd` file (e.g., adding/deleting users)

    Can `root` *decrypt passwords? Is it obvious why a user should not be able to edit the password file?*

- What `root` cannot do
  - Change a read-only file system
  - Decrypt passwords
  - The list is very short...

Providing `root` Privileges

- Perhaps it is necessary to give users root privileges
  - Give the user the root password, then they can use `su`
  - `su` stands for *switch user*, syntax is `su` *userName*

Terminal

```
> su
Password:
#
```

Any problems with this approach?

- `sudo` provides a more control method to share `root`
  - `sudo` stands for *superuser do*
  - Prompts for a personal password and confirms the request

`sudo` Permissions

- Command syntax is `sudo` *command*

Terminal

```
> sudo more /etc/shadow
Password:
```

- `sudo` will check the `/etc/sudoers` file for the permission

```
# /etc/sudoer
#
# This file MUST be edited with the 'visudo' command as root.
...
# User privilege specification
root      ALL=(ALL) ALL
...
# Members of the admin group may gain root privileges
%admin    ALL=(ALL) ALL
```

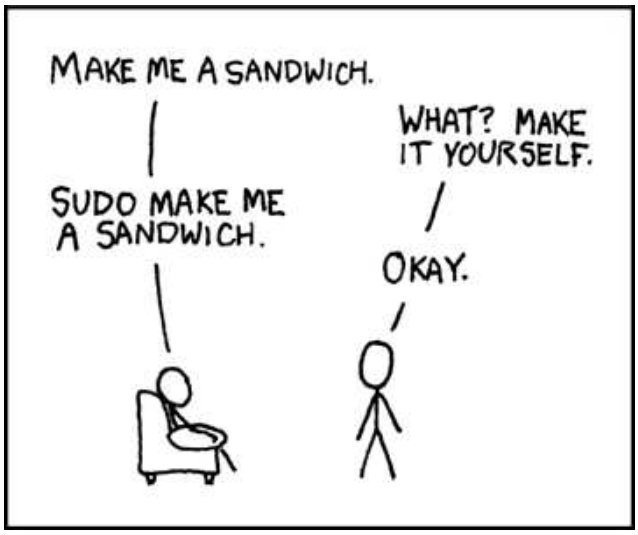
- Lines in the file have the format
  userList hostList = operatorList tagList commandList
- For example

  `pluf ALL= NOPASSWD: /sbin/shutdown`
  - Gives `pluf` permission to execute the `shutdown` command

    `root ALL=(ALL) ALL`
  - Gives `root` permission to do anything as any user (**should have**)

    `%admin ALL=(ALL) ALL`
  - Gives any member of the `admin` group (`/etc/groups`) root privileges, requires the users to enter their password, not `root`'s (*awesome*)
- Use the command `visudo` to edit the file...

Now You Get It, *Right?*



Controlled Invocation

- Unix requires superuser privilege to perform certain tasks
  - Users may occasionally need such privileges

    What is an example?

  - However normal users do not have such status (permissions)
- Unix has a *work-around* to allow this to happen
  - Set User ID (SUID) and Set Group ID (SGID) programs
  - Programs that are SUID give a user a different status during their execution, then return to *normal* once complete
- Example SUID programs (Unix commands) include

`passwd`	change user password
`login`	login program
`su`	change UID program

- Following command will/may provide a list from `/usr/bin`

Terminal

```
> find /usr/bin -xdev \( -perm -4000 \) -type f -print0 | xargs -0 ls -l
-r-sr-xr-x 1 root wheel 80368 Aug 23 2011 /usr/bin/crontab
-r-s--x--x 1 root wheel 51664 Feb 6 2012 /usr/bin/ipcs
-r-sr-xr-x 1 root wheel 68224 Feb 6 2012 /usr/bin/login
-r-sr-xr-x 1 root wheel 44000 Feb 6 2012 /usr/bin/newgrp
-r-sr-xr-x 1 root wheel 44304 Aug 23 2011 /usr/bin/quota
-r-sr-xr-x 1 root wheel 53520 Aug 23 2011 /usr/bin/rlogin
-r-sr-xr-x 1 root wheel 44352 Aug 23 2011 /usr/bin/rsh
-r-sr-xr-x 1 root wheel 58176 Aug 33 2011 /usr/bin/su
-r-s--x--x 1 root wheel 327920 Feb 6 2012 /usr/bin/sudo
-r-sr-xr-x 1 root wheel 189808 Aug 23 2011 /usr/bin/top
```

- You can also write your own `setuid()` program

- A program that sets the UID could look like

```
#include<unistd.h>
// standard C/C++ program stuff here
...
int userID = getuid();  // gets user's current UID
setuid(0);              // set UID to root
// do something only root can do...
setuid(userID);         // once complete, restore original UID
```

- SUID programs are extremely dangerous
  - If you alter what happens during the time the program is `root`...
  - A common objective is to *spawn a shell* during this time
  - However, how can you alter a programs execution...
  - This is the purpose of *smashing the stack*

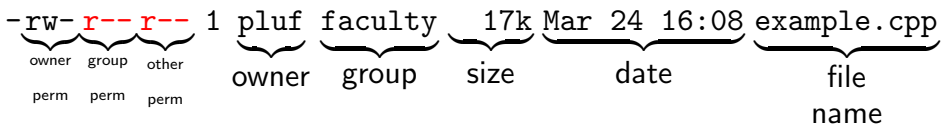
Unix File Structure

- Unix arranges files in a tree structure
  - Base of the tree is the *root* directory /
  - System consists of files and directories
  - Every directory contains two files `.` and `..`
  - Every file has an *owner* and belongs to a *group*
- The command `ls -al` gives a listing of the current directory

Terminal

```
> ls -al
drwxr-xr-x   2  pluf faculty   4.0k Mar 24 15:58 ./
drwxr-xr-x  16  pluf faculty   4.0k Mar 24 14:47 ../
-rw-r--r--   1  pluf faculty    19k Mar 24 16:08 example.cpp
```

- For each file listed



  - First character indicates if the file is a directory `d`
  - Next 3 characters indicate the read `r`, write `w`, and execute `x` permissions for the owner
  - Next 3 characters indicate the `rwx` permissions for the group
  - Next 3 characters indicate the `rwx` permissions for others
  - If the permission is not *granted* then a `-` is displayed

- If the file is a SUID then the execute permission of the owner is `s`

Terminal

```
> ls -al /usr/bin/passwd
-r-s--x--x   1  root  root    12k Jan 2000 /usr/bin/passwd*
```

- Permissions can be changed using `chmod` command
  - Add permission syntax is `chmod` *who+perm fileName*
  - Remove permission syntax is `chmod` *who-perm fileName*

Terminal

```
> chmod o-r example.cpp
> chmod g+w example.cpp
> chmod u+s vulnerableExecutable
```

File `stat`

- *As a side note*, the `stat` command will show more about a file

Terminal

```
> stat /home/fileName.txt
File:  '/home/fileName.txt'
Size:  887       Blocks: 8      IO Block:  4096  regular file
Device:  801h/2049d Inode:  392918  Links:  1
Access:  (0680/-rw-r----)  Uid:  (0/root)   Gid:  (48/shadow)
Access:  2011-01-26 11:28:18.00000000 -0500
Modify:  2011-01-26 11:20:58.00000000 -0500
Change:  2011-01-26 11:20:58.000 00000 -0500
```

  - `Device` is the device number in hex and decimal
  - `Access` is the last access time of the file
  - `Modify` is the last modification time of the file
  - `Change` is the last change time of the inode data of that file.

Environment Variables

- Environment variables are *hidden* input
  - Affect program behavior, so ignoring how they are set is dangerous

Terminal

```
> printenv
SHELL=/bin/bash
USER=fulp
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/
HOME=/user/pluf
LOGNAME=pluf
...
```

- When executed, a program may use these variables for system access
  - Variables include `PATH`, `IFS`, `LD_LIBRARY_PATH`, and `LD_PRELOAD`

Search Paths

- Users interact with the OS through a *shell*
  - Shell is a command line interpreter
  - Examples include `tcsh`, `bash`, and `csh`
- As a matter of convenience users can type run a program (command) without specifying the complete path
  - For example you can type `ls` instead of `/bin/ls`
  - The shell knows where to look using a `PATH` variable
- The `PATH` environment variable lists the directories to search
  - Defined in your `.cshrc` file located in your home directory
  - **Order matters**

    ```
    PATH=.:$HOME/bin:/bin/:/usr/bin:/usr/local
    ```


Path Attack

Assume system administrator has `.` at the beginning of their `PATH`

1. You create a file called `ls` with the following contents

   ```
   #!/bin/csh
   cp /bin/sh/ ./Stuff/sume
   chmod 4555 ./Stuff/sume
   rm -f $0
   exec /bin/ls $1
   ```

2. Create another unreadable file (contents are not important)
3. Then you issue the commands `chmod 700 .; touch ./-f`
4. You tell the admin you cannot erase one of your files (`-f`)
5. Admin becomes `root` moves to your directory and enters `ls`
   - But your version of `ls` is executed
6. Congratulations, you now have `root`

IFS

- IFS environment variable stands for Internal Field Separators
  - Identifies characters to be interpreted as whitespace
- If these libraries are exchanged with malware

Terminal

```
> IFS="/ \t\n"; export IFS
> PATH=".:$PATH"
```

- Start a program which uses an absolute PATH from a Bourne shell

Intended	Actually Used
`system("/bin/mail root");`	`system(" bin mail root");`

  - It will attempt to execute a command called bin in the current directory of the user

LD_LIBRARY_PATH

- Used when searching for dynamic libraries
  - Virtually every Unix program depends on libc.so and Windows program relies on DLL's
- If these libraries are exchanged with malware

Terminal

```
> setenv LD_LIBRARY_PATH /tmp:$LD_LIBRARY_PATH
```

- Given the above, /tmp will be searched first
  - So place the hacked version of libc.so in /tmp
- Most modern C runtime libraries have fixed this problem by ignoring the LD_LIBRARY_PATH variable when the EUID is not equal to the UID

LD_PRELOAD

- Many UNIX systems allow you to *pre-load* shared libraries
  - Done using `LD_LIBRARY_PATH` environment variable
  - Allows you to do interesting things like replace standard C library functions or even the C interfaces to system calls with your own functions
- If these libraries are exchanged with malware

Terminal

```
> cc -o malloc_interposer.so -G -Kpic malloc_interposer.c
> mv malloc_interposer.so /tmp
> setenv LD_PRELOAD /tmp/malloc_interposer.so
```


vi + PATH + IFS = BFF

- Consider using `vi` to edit a file
  1. `vi` a file
  2. Kill `vi` without saving file
  3. `vi` invokes `expreserve` which will save buffer
  4. `expreserve` uses `mail` to email user
- Details of what happens in this case
  - `expreserve` is a `setuid`, so `mail` is called with root privilege
  - Uses `system("mail user");` or `system("/bin/mail user");`
- Attack method
  - Change `PATH`, `IFS` `IFS="/binal\t\n"` causes `m` be invoked, instead of `"/bin/mail"`

How to Avoid Environment Variables?

- Possible to avoid environment variables
  - Set external pointer variable `environ` to zero (null)

```
extern char **environ;
int main(int argc, char* argv[])
{
   environ = 0;
}
```

Is this approach acceptable in all cases?

Root Processes

> *"Init scripts are the scripts located in /etc/init.d. These scripts are part of the bootup sequence of Ubuntu. During boot, they are not called directly, but through a structure of symbolic links which manage the services which are to be started in a particular runlevel. The scripts which are symlinked from /etc/rcS.d are executed first. Then the scripts in /etc/rcN.d/ are executed, with N being the chosen runlevel (default 2). "*

- Initial boot `/etc/rc*`
  - List of system processes started at boot

    Do they execute as user, `root`, ... does it matter?

- Network processes `/etc/inetd.conf`
  - List of network services started at boot

- Command ps or top will show what is running

Terminal

```
> ps
PID  TT  STAT      TIME COMMAND
518  p1  S      0:00.04 -bash
670  p2  S+     0:00.09 vi areYouReallyReadingThis.txt
```

- ps -ef will list the Parent Process ID (PPID)

Terminal

```
> ps -ef
UID      PID  PPID  C STIME TTY          TIME CMD
root      15     1    0 11:24 ?        00:00:00 /tmp/moniker
root      242   15    0 11:25 ?        00:00:00 /tmp/cr8
root      244    1    0 11:25 ?        00:00:00 /tmp/rucc
```

So what? How does that help?

An Attempt to Make Linux Secure-er

- Set a password to disallow booting from a floppy, CD, etc...
- Disable *special* and *default* accounts
  - lp, sync, shutdown, operator, ...
  - This also applies to applications *Such as?*
- Set a timeout (TMOUT=) in the profile file (users and root)
- Disable console-equivalent user access to reboot, shutdown, ...
- Disable unnecessary network services (*good luck...*)
  - Edit the /etc/inetd.conf file
  - Make certain root is the owner of /etc/inetd.conf
  - Change permissions of /etc/inetd.conf to 600 *Why?*

- Use `TCP_WRAPPERS` and/or firewall to control access
- Do not display the *system issue* information at log in

  What?

- Review the `/etc/host.conf` file
  - This file resolves IP names to addresses
  - You can make it ask DNS first or have the file resolve

    Which is better?

- Immunize the `/etc/services` file
  - Prevent unauthorized deletion or addition of services
  - Use the command `chattr +i /etc/services`

- Make signatures of any setuid programs

  What? How?

- Prevent users from `su`

  Why? How can you access `root`*?*

- Have the shell erase the command history of a user at log out
- Disable the Control-Alt-Delete keyboard shutdown command
- Fix permissions under `/etc/rc.d/init.d` directory for script files
  - Use the command `chmod -R 700 /etc/rc.d/init.d/*`

    What is the result?

- Turn-off the machine and don't turn it on... ever

More About Services

- Turn off, and perhaps uninstall, all unnecessary services
- Make sure that any services that are installed are updated
  - *Every server application has potential exploits...*
- Limit connections to us from outside sources
  - Allow only the minimum traffic necessary

one of the interesting aspects of Linux, is the different distributions like Caldera, Red Hat, SuSE, and Debian. While these are all "Linux", and may share certain features, there is surely some differences as to what utilities they may install as defaults. Most Linux distributions will write their own system configuration tools as well. And with Linux, there is always more than one way to skin a cat.

System Audit

- *So what is running on our system anyway?*

Terminal

```
> netstat -tap | grep "LISTEN"
*:telnet           *:*  LISTEN  988/inetd
*:finger           *:*  LISTEN  988/inetd
*:x11              *:*  LISTEN  978/x11
*:http             *:*  LISTEN  1590/http
*:ftp              *:*  LISTEN  988/inetd
*:1694             *:*  LISTEN  1319/rpc.mountd
```

- Lists the services waiting for a connection
  - Time for an awkward moment...

Things Not Necessary

- NFS (Network File System) and related services
  - Unix service for sharing file systems across a network
  - Great system for LAN usage, but dangerous over the Internet
- `rpc.*` services
  - Remote procedure calls, typically NFS and NIS related
- Printer services (`lpd`)
  - Are you a print server, perhaps a unicorn?
- The so-called r* (`rsh`, `rlogin`, `rexec`, `rcp`, etc...) services
  - Unnecessary, insecure and potentially dangerous
  - Better utilities are available if these are needed.
  - `ssh` *will do everything these command do, and in a much more sane way*

- `telnet` and `ftp` servers
  - Few reasons for these anymore, use `sshd` and `sftp`
  - `ftp` *is a proper protocol only for someone who is running a dedicated ftp server, and who has the time and skill to keep it buttoned down. For everyone else, it is potentially big trouble*
- BIND (`named`), a DNS server package.
  - Can be used, but is not necessary in many situations
  - *if you are providing DNS look ups for domains to the rest of the world. If you are not sure what this means, you do not need*
- Mail Transport Agents (`sendmail`, `exim`, `postfix`, `qmail`)
  - Most single computers will not really need this if you are do not directly receiving mail from Internet hosts

How to Stop Services

- Services can be started different ways and places
  - This is determine how it is stopped
- System services are typically either started by
  - `init` scripts, location depends on distro
  - `inetd` (or its replacement `xinetd`) on most distributions
- Of course we can `kill` the process using the PID

  But... ?

`init` Services

- Started automatically during the boot or runlevel change
  - Located in `/etc/init.d/` (or possibly `/etc/rc.d/init.d/`)
- To list services

Terminal

```
> ls -l /etc/init.d/ | more
```

- To stop a service

Terminal

```
> /etc/init.d/SERVICE_NAME stop
```

How long does it stop the service?

inetd Services

- Called a *super-daemon* because it spawns sub-daemons
  - inetd generally started via init scripts
  - *listens* for ports to enable in config file, /etc/inetd.conf
  - Services listed in the config file are controled by inetd
  - *Also shown in the netstat command...*
- inetd.conf file contents

```
+-----------------------------------------+
#
# inetd.conf This file describes the ...
:
#echo stream tcp nowait root internal
pop-3 stream tcp nowait root /usr/sbin/tcpd ipop3d
:
```

- Comment lines begin with #

- Can check for services using grep

Terminal

```
> grep -v '^#' /etc/inetd.conf
pop-3 stream tcp nowait root /usr/sbin/tcpd ipop3d
```

- Once you have made changes, restart

Terminal

```
> /etc/init.d/inetd restart
```

- xinetd is an inetd replacement with enhancements. It essentially
  - Serves the same purpose as inetd
  - Configuration is different, can be in the file /etc/xinetd.conf or individual files in /etc/xinetd.d/

Audit Trails

- Things that are *typically* logged
  - Crashes, login/logout, `su`, login failures, dialouts printer use/errors, `mail`/`www`, firewall exceptions
- Things that are not *typically* logged
  - Failed file access, some network accesses (`rcp`), file changes, password changes, `www` access failures, and superuser actions

So what, what is the advantage of log files?

What is the disadvantage of log files?

`syslog`

- Logging is done with `syslog()`
  - Writes to the `syslogd` daemon, also has UDP support

    What is UDP? Why have UDP?
- The file `/etc/syslog.conf` indicates log locations

```
# Log all kernel messages to the console
kern.*                      /dev/console
# Log all mail messages in one place
mail.*                      /var/log/maillog
# Everyone gets emergence messages
*.emerge                     *
```

  - First entry on a line is the *selector*
  - Second entry is the *action*
- *Now we are considering Intrusion Detection Systems...*

Configurations Guidelines and Standards

- There are some configuration standards/baselines
  - United States Government Configuration Baseline (USGCB)
  - DoD DISA Security Technical Implementation Guides (STIGs)
- Provide recommendations (requirements) for software configurations
  - Often include Security Content Automation Protocol (SCAP) content that allows automated evaluation and maintenance
- SCAP consists of
  - Common Vulnerabilities and Exposures (CVE)
  - Common Configuration Enumeration (CCE)
  - Common Platform Enumeration (CPE)
  - Common Vulnerability Scoring System (CVSS)
  - Extensible Configuration Checklist Description Format (XCCDF)
  - Open Vulnerability and Assessment Language (OVAL)

DoD Information Assurance Support Environment

- Security Technical Implementation Guides (STIGs)
  - Configuration standards for DOD Information Assurance (IA)
  - Guidance to "lock down" software that might be vulnerable
  - Available at `http://www.stigviewer.com`
- STIG Security Checklist contains instructions or procedures to manually verify compliance (for example, the following for RHEL5)

Group ID (Vulid): V-797
Group Title: GEN001400
Rule ID: SV-37361r1_rule
Severity: CAT II
Rule Version (STIG-ID): GEN001400
Rule Title: The /etc/shadow (or equivalent) file must be owned by root.
Vulnerability Discussion: The `/etc/shadow` file contains the list of local system accounts. It is vital to system security and must be protected from unauthorized modification. Failure to give ownership of sensitive files or utilities to root or bin provides the designated owner and unauthorized users with the potential to access sensitive information or change the system configuration which could weaken the system's security posture.
Responsibility: System Administrator
IAControls: ECLP-1
Check Content:
Check the ownership of the texttt/etc/shadow file.
`# ls -lL /etc/shadow`
If the `/etc/shadow` file is not owned by root, this is a finding.
Fix Text: Change the ownership of the `/etc/shadow` (or equivalent) file. `# chown root /etc/shadow`
CCI: CCI-000225

Group ID (Vulid): V-800
Group Title: GEN001420
Rule ID: SV-37368r1_rule
Severity: CAT II
Rule Version (STIG-ID): GEN001400
Rule Title: The `/etc/shadow` (or equivalent) file must have mode 0400.
Vulnerability Discussion: The `/etc/shadow` file contains the list of local system accounts. It is vital to system security and must be protected from unauthorized modification. The file also contains password hashes which must not be accessible to users other than root.
Responsibility: System Administrator
IAControls: ECLP-1
Check Content:
Check the ownership of the /etc/shadow file.
```
# ls -lL /etc/shadow
```
If the `/etc/shadow` file has a mode more permissive than 0400, this is a finding.
Fix Text: Change the mode of the `/etc/shadow` (or equivalent) file. `# chmod 0400 /etc/shadow`
CCI: CCI-000225

Group ID (Vulid): V-900
Group Title: GEN001460
Rule ID: SV-37379r1_rule
Severity: CAT III
Rule Version (STIG-ID): GEN001460
Rule Title: All interactive user home directories defined in the `/etc/passwd` file must exist.
Vulnerability Discussion: If a user has a home directory defined that does not exist, the user may be given the / directory, by default, as the current working directory upon logon. This could create a Denial of Service because the user would not be able to perform useful tasks in this location.
IAControls: ECSC-1
Check Content:
Use pwck to verify assigned home directories exist.
```
# pwck
```
If any user's assigned home directory does not exist, this is a finding.
Fix Text: If a user has no home directory, determine why. If possible, delete accounts without a home directory. If the account is valid, then create the home directory using the appropriate system administration utility or manually. For instance: `mkdir directoryname`; copy the skeleton files into the directory; `chown accountname` for the new directory and the skeleton files. Document all changes.
CCI: CCI-000225