Linux Shadow Password HOWTO

Table of Contents

Table of Contents

Linux Shadow Password HOWTO

Michael H. Jackson, `mhjack@tscnet.com`

v1.3, 3 April 1996

This document aims to describe how to obtain, install, and configure the Linux password Shadow Suite. It also discusses obtaining, and re]installing other software and network daemons that require access to user passwords. This other software is not actually part of the Shadow Suite, but these programs will need to be recompiled to support the Shadow Suite. This document also contains a programming example for adding shadow support to a program. Answers to some of the more frequently asked questions are included near the end of this document.

9.Frequently Asked Questions.

10.Copyright Message.

11.Miscellaneous and Acknowledgments.

1.Introduction.

This is the Linux Shadow−Password−HOWTO. This document describes why and how to add shadow password support on a Linux system. Some examples of how to use some of the *Shadow Suite's* features is also included.

When installing the *Shadow Suite* and when using many of the utility programs, you must be logged in as *root*. When installing the *Shadow Suite* you will be making changes to system software, and it is highly recommended that you make backup copies of programs as indicated. I also recommend that you read and understand all the instructions before you begin.

1.1 Changes from the previous release.

```
Additions:
        Added a sub−section on why you might not want to install shadow
        Added a sub−section on updating the xdm program
        Added a section on how to put Shadow Suite features to work
        Added a section containing frequently asked questions

Corrections/Updates:
        Corrected html references on Sunsite
        Corrected section on wu−ftp to reflect adding −lshadow to the Makefile
        Corrected minor spelling and verbiage errors
        Changed section on wu−ftpd to support ELF
        Updated to reflect security problems in various login programs
        Updated to recommend the Linux Shadow Suite by Marek Michalkiewicz
```

1.2 New versions of this document.

The latest released version of this document can always be retrieved by anonymous FTP from:

sunsite.unc.edu

`/pub/Linux/docs/HOWTO/Shadow-Password-HOWTO`
or:

`/pub/Linux/docs/HOWTO/other-formats/Shadow-Password-HOWTO{-html.tar,ps,dvi}.gz`

or via the World Wide Web from the Linux Documentation Project Web Server, at page: Shadow−Password−HOWTO or directly from me, `<mhjack@tscnet.com>`. It will also be posted to the newsgroup: `comp.os.linux.answers`

This document is now packaged with the Shadow−YYDDMM packages.

1.3 Feedback.

Please send any comments, updates, or suggestions to me: Michael H. Jackson <mhjack@tscnet.com> The sooner I get feedback, the sooner I can update and correct this document. If you find any problems with it, please mail me directly as I very rarely stay up−to−date on the newsgroups.

2.Why shadow your passwd file?

By default, most current Linux distributions do not contain the *Shadow Suite* installed. This includes Slackware 2.3, Slackware 3.0, and other popular distributions. One of the reasons for this is that the copyright notices in the original *Shadow Suite* were not clear on redistribution if a fee was charged. Linux uses a GNU Copyright (sometimes refereed to as a Copyleft) that allows people to package it into a convenient package (like a CD−ROM distribution) and charge a fee for it.

The current maintainer of the *Shadow Suite*, Marek Michalkiewicz <marekm@i17linuxb.ists.pwr.wroc.pl> received the source code from the original author under a BSD style copyright that allowed redistribution. Now that the copyright issues are resolved, it is expected that future distributions will contain password shadowing by default. Until then, you will need to install it yourself.

If you installed your distribution from a CD−ROM, you may find that, even though the distribution did not have the *Shadow Suite* installed, some of the files you need to install the *Shadow Suite* may be on the CD−ROM.

However, Shadow Suite versions 3.3.1, 3.3.1−2, and shadow−mk all have security problems with their login program and several other suid root programs that came with them, and should no longer be used.

All of the necessary files may be obtained via anonymous FTP or through the World Wide Web.

On a Linux system without the *Shadow Suite* installed, user information including passwords is stored in the `/etc/passwd` file. The password is stored in an *encrypted* format. If you ask a cryptography expert, however, he or she will tell you that the password is actually in an *encoded* rather than *encrypted* format because when using crypt(3), the text is set to null and the password is the key. Therefore, from here on, I will use the term *encoded* in this document.

The algorithm used to encode the password field is technically referred to as a *one way hash function*. This is an algorithm that is easy to compute in one direction, but very difficult to calculate in the reverse direction. More about the actual algorithm used can be found in section 2.4 or your crypt(3) manual page.

When a user picks or is assigned a password, it is encoded with a randomly generated value called the *salt*. This means that any particular password could be stored in 4096 different ways. The *salt* value is then stored with the encoded password.

When a user logs in and supplies a password, the *salt* is first retrieved from the stored encoded password. Then the supplied password is *encoded* with the *salt* value, and then compared with the *encoded* password. If there is a match, then the user is authenticated.

It is computationally difficult (but not impossible) to take a randomly *encoded* password and recover the original password. However, on any system with more than just a few users, at least some of the passwords will be common words (or simple variations of common words).

System crackers know all this, and will simply encrypt a dictionary of words and common passwords using all possible 4096 *salt* values. Then they will compare the encoded passwords in your `/etc/passwd` file with their database. Once they have found a match, they have the password for another account. This is referred to as a *dictionary attack*, and is one of the most common methods for gaining or expanding unauthorized access to a system.

If you think about it, an 8 character password encodes to 4096 * 13 character strings. So a dictionary of say 400,000 common words, names, passwords, and simple variations would easily fit on a 4GB hard drive. The attacker need only sort them, and then check for matches. Since a 4GB hard drive can be had for under $1000.00, this is well within the means of most system crackers.

Also, if a cracker obtains your `/etc/passwd` file first, they only need to encode the dictionary with the `salt` values actually contained in your `/etc/passwd` file. This method is usable by your average teenager with a couple of hundred spare Megabytes and a 486 class computer.

Even without lots of drive space, utilities like crack(1) can usually break at least a couple of passwords on a system with enough users (assuming the users of the system are allowed to pick their own passwords).

The `/etc/passwd` file also contains information like user ID's and group ID's that are used by many system programs. Therefore, the `/etc/passwd` file *must* remain world readable. If you were to change the `/etc/passwd` file so that nobody can read it, the first thing that you would notice is that the `ls -l` command now displays user ID's instead of names!

The *Shadow Suite* solves the problem by relocating the passwords to another file (usually `/etc/shadow`). The `/etc/shadow` file is set so that it cannot be read by just anyone. Only *root* will be able to read and write to the `/etc/shadow` file. Some programs (like xlock) don't need to be able to change passwords, they only need to be able to verify them. These programs can either be run *suid root* or you can set up a group *shadow* that is allowed read only access to the `/etc/shadow` file. Then the program can be run *sgid shadow*.

By moving the passwords to the `/etc/shadow` file, we are effectively keeping the attacker from having access to the encoded passwords with which to perform a *dictionary attack*.

Additionally, the *Shadow Suite* adds lots of other nice features:

- A configuration file to set login defaults (`/etc/login.defs`)
- Utilities for adding, modifying, and deleting user accounts and groups
- Password aging and expiration
- Account expiration and locking
- Shadowed group passwords (optional)
- Double length passwords (16 character passwords) NOT RECOMMENDED]
- Better control over user's password selection
- Dial–up passwords
- Secondary authentication programs [NOT RECOMMENDED]

Installing the *Shadow Suite* contributes toward a more secure system, but there are many other things that can also be done to improve the security of a Linux system, and there will eventually be a series of Linux Security HOWTO's that will discuss other security measures and related issues.

For current information on other Linux security issues, including warnings on known vulnerabilities see the Linux Security home page.

2.1 Why you might NOT want to shadow your passwd file.

There are a few circumstances and configurations in which installing the *Shadow Suite* would *NOT* be a good idea:

- The machine does not contain user accounts.
- Your machine is running on a LAN and is using NIS (Network Information Services) to get or supply user names and passwords to other machines on the network. (This can actually be done, but is beyond the scope of this document, and really won't increase security much anyway)
- Your machine is being used by terminal servers to verify users via NFS (Network File System), NIS, or some other method.
- Your machine runs other software that validates users, and there is no shadow version available, and you don't have the source code.

2.2 Format of the /etc/passwd file

A non−shadowed `/etc/passwd` file has the following format:

```
username:passwd:UID:GID:full_name:directory:shell
```

Where:

`username`

The user (login) name

`passwd`

The encoded password

`UID`

Numerical user ID

`GID`

Numerical default group ID

`full_name`

The user's full name − Actually this field is called the GECOS (General Electric Comprehensive Operating System) field and can store information other than just the full name. The Shadow commands and manual pages refer to this field as the comment field.

`directory`

User's home directory (Full pathname)

shell

User's login shell (Full Pathname)

For example:

```
username:Npge08pfz4wuk:503:100:Full Name:/home/username:/bin/sh
```

Where `Np` is the salt and `ge08pfz4wuk` is the *encoded* password. The encoded salt/password could just as easily have been `kbeMVnZM0oL7I` and the two are exactly the same password. There are 4096 possible encodings for the same password. (The example password in this case is 'password', a really *bad* password).

Once the shadow suite is installed, the `/etc/passwd` file would instead contain:

```
username:x:503:100:Full Name:/home/username:/bin/sh
```

The `x` in the second field in this case is now just a place holder. The format of the `/etc/passwd` file really didn't change, it just no longer contains the *encoded* password. This means that any program that reads the `/etc/passwd` file but does not actually need to verify passwords will still operate correctly.

The passwords are now relocated to the shadow file (usually `/etc/shadow` file).

2.3 Format of the shadow file

The `/etc/shadow` file contains the following information:

```
username:passwd:last:may:must:warn:expire:disable:reserved
```

Where:

username

The User Name

passwd

The Encoded password

last

Days since Jan 1, 1970 that password was last changed

may

Days before password may be changed

must

Days after which password must be changed

warn

Days before password is to expire that user is warned

expire

Days after password expires that account is disabled

disable

Days since Jan 1, 1970 that account is disabled

reserved

A reserved field

The previous example might then be:

```
username:Npge08pfz4wuk:9479:0:10000::::
```

2.4 Review of crypt(3).

From the crypt(3) manual page:

"*crypt* is the password encryption function. It is based on the *Data Encryption Standard* algorithm with variations intended (among other things) to discourage use of hardware implementations of a key search.

[The] key is a user's typed password. [The encoded string is all NULLs]

[The] *salt* is a two−character string chosen from the set [a−zA−Z0−9./]. This string is used to perturb the algorithm in one of 4096 different ways.

By taking the lowest 7 bit[s] of each character of the key, a 56−bit key is obtained. This 56−bit key is used to encrypt repeatedly a constant string (usually a string consisting of all zeros). The returned value points to the encrypted password, a series of 13 printable ASCII characters (the first two characters represent the salt itself). The return value points to static data whose content is overwritten by each call.

Warning: The key space consists of 2**56 equal 7.2e16 possible values. Exhaustive searches of this key space **are possible** using massively parallel computers. Software, such as `crack(1)`, is available which will search the portion of this key space that is generally used by humans for passwords. Hence, password selection should, at minimum, avoid common words and names. The use of a `passwd(1)` program that checks for crackable passwords during the selection process is recommended.

The DES algorithm itself has a few quirks which make the use of the `crypt(3)` interface a very poor choice for anything other than password authentication. If you are planning on using the `crypt(3)` interface for a cryptography project, don't do it: get a good book on encryption and one of the widely available DES libraries."

Most *Shadow Suites* contain code for doubling the length of the password to 16 characters. Experts in `des` recommend against this, as the encoding is simply applied first to the left half and then to the right half of the longer password. Because of the way `crypt` works, this may make for a *less* secure encoded password then if double length passwords were not used in the first place. Additionally, it is less likely that a user will be able to remember a 16 character password.

There is development work under way that would allow the authentication algorithm to be replaced with something more secure and with support for longer passwords (specifically the MD5 algorithm) and retain compatibility with the `crypt` method.

If you are looking for a good book on encryption, I recommend:

```
"Applied Cryptography: Protocols, Algorithms, and Source Code in C"
by Bruce Schneier <schneier@chinet.com>
ISBN: 0-471-59756-2
```

3.Getting the Shadow Suite.

3.1 History of the Shadow Suite for Linux

DO NOT USE THE PACKAGES IN THIS SECTION, THEY HAVE SECURITY PROBLEMS

The original *Shadow Suite* was written by `John F. Haugh II`.

There are several versions that have been used on Linux systems:

- `shadow-3.3.1` is the original.
- `shadow-3.3.1-2` is Linux specific patch made by Florian La Roche <flla@stud.uni–sb.de> and contains some further enhancements.
- `shadow-mk` was specifically packaged for Linux.

The `shadow-mk` package contains the `shadow-3.3.1` package distributed by `John F. Haugh II` with the `shadow-3.3.1-2 patch` installed, a few fixes made by Mohan Kokal <magnus@texas.net> that make installation a lot easier, a patch by `Joseph R.M. Zbiciak` for `login1.c` (login.secure) that eliminates the –f, –h security holes in /bin/login, and some other miscellaneous patches.

The `shadow.mk` package was the *previously* recommended package, but should be replaced due to a *security problem* with the `login` program.

There are *security problems* with Shadow versions 3.3.1, 3.3.1−2, and shadow−mk involving the `login` program. This `login` bug involves not checking the length of a login name. This causes the buffer to overflow causing crashes or worse. It has been rumored that this buffer overflow can allow someone with an account on the system to use this bug and the shared libraries to gain *root* access. I won't discuss exactly how this is possible because there are a lot of Linux systems that are affected, but systems with these *Shadow Suites* installed, and most pre−ELF distributions *without* the *Shadow Suite* are vulnerable!

For more information on this and other Linux security issues, see the Linux Security home page (Shared Libraries and login Program Vulnerability)

3.2 Where to get the Shadow Suite.

The only recommended *Shadow Suite* is still in BETA testing, however the latest versions are safe in a production environment and don't contain a vulnerable `login` program.

The package uses the following naming convention:

```
shadow-YYMMDD.tar.gz
```

where `YYMMDD` is the issue date of the Suite.

This version will eventually be *Version 3.3.3* when it is released from Beta testing, and is maintained by Marek Michalkiewicz <marekm@i17linuxb.ists.pwr.wroc.pl>. It's available as: shadow−current.tar.gz.

The following mirror sites have also been established:

- ftp://ftp.icm.edu.pl/pub/Linux/shadow/shadow−current.tar.gz
- ftp://iguana.hut.fi/pub/linux/shadow/shadow−current.tar.gz
- ftp://ftp.cin.net/usr/ggallag/shadow/shadow−current.tar.gz
- ftp://ftp.netural.com/pub/linux/shadow/shadow−current.tar.gz

You should use the currently available version.

You should NOT use a version *older* than `shadow-960129` as they also have the `login` security problem discussed above.

When this document refers to the *Shadow Suite* I am referring to the this package. It is assumed that this is the package that you are using.

For reference, I used `shadow-960129` to make these installation instructions.

If you were previously using `shadow-mk`, you should upgrade to this version and rebuild everything that you originally compiled.

3.3 What is included with the Shadow Suite.

The *Shadow Suite* contains replacement programs for:

`su, login, passwd, newgrp, chfn, chsh, and id`

The package also contains the new programs:

`chage, newusers, dpasswd, gpasswd, useradd, userdel, usermod, groupadd, groupdel, groupmod, groups, pwck, grpck, lastlog, pwconv, and pwunconv`

Additionally, the library: `libshadow.a` is included for writing and/or compiling programs that need to access user passwords.

Also, manual pages for the programs are also included.

There is also a configuration file for the login program which will be installed as `/etc/login.defs`.

4.Compiling the programs.

4.1 Unpacking the archive.

The first step after retrieving the package is unpacking it. The package is in the tar (tape archive) format and compressed using gzip, so first move it to `/usr/src`, then type:

```
tar -xzvf shadow-current.tar.gz
```

This will unpack it into the directory: `/usr/src/shadow-YYMMDD`

4.2 Configuring with the config.h file

The first thing that you need to do is to copy over the `Makefile` and the `config.h` file:

```
cd /usr/src/shadow-YYMMDD
cp Makefile.linux Makefile
cp config.h.linux config.h
```

You should then take a look at the `config.h` file. This file contains definitions for some of the configuration options. If you are using the *recommended* package, I recommend that you disable group shadow support for your first time around.

By default shadowed group passwords are enabled. To disable these edit the `config.h` file, and change the `#define SHADOWGRP` to `#undef SHADOWGRP`. I recommend that you disable them to start with, and then if you really want group passwords and group administrators that you enable it later and recompile. If you leave it enabled, you *must* create the file `/etc/gshadow`.

Enabling the long passwords option is NOT recommended as discussed above.

Do NOT change the setting: `#undef AUTOSHADOW`

The `AUTOSHADOW` option was originally designed so that programs that were shadow ignorant would still function. This sounds good in theory, but does not work correctly. If you enable this option, and the program runs as root, it may call `getpwnam()` as root, and later write the modified entry back to the `/etc/passwd` file (with the *no−longer−shadowed password*). Such programs include chfn and chsh. (You can't get around this by swapping real and effective uid before calling `getpwnam()` because root may use chfn and chsh too.)

The same warning is also valid if you are building libc, it has a `SHADOW_COMPAT` option which does the same thing. It *should NOT* be used! If you start getting encoded passwords back in your `/etc/passwd` file, this is the problem.

If you are using a `libc` version prior to 4.6.27, you will need to make a couple more changes to `config.h` and the `Makefile`. To `config.h` edit and change:

```
#define HAVE_BASENAME
```

to:

```
#undef HAVE_BASENAME
```

And then in the `Makefile`, change:

```
SOBJS = smain.o env.o entry.o susetup.o shell.o \
        sub.o mail.o motd.o sulog.o age.o tz.o hushed.o

SSRCS = smain.c env.c entry.c setup.c shell.c \
        pwent.c sub.c mail.c motd.c sulog.c shadow.c age.c pwpack.c rad64.c \
        tz.c hushed.c
```

```
SOBJS = smain.o env.o entry.o susetup.o shell.o \
        sub.o mail.o motd.o sulog.o age.o tz.o hushed.o basename.o

SSRCS = smain.c env.c entry.c setup.c shell.c \
        pwent.c sub.c mail.c motd.c sulog.c shadow.c age.c pwpack.c rad64.c \
        tz.c hushed.c basename.c
```

These changes add the code contained in `basename.c` which is contained in `libc 4.6.27` and later.

4.3 Making backup copies of your original programs.

It would also be a good idea to track down and make backup copies of the programs that the shadow suite will replace. On a Slackware 3.0 system these are:

- /bin/su
- /bin/login
- /usr/bin/passwd
- /usr/bin/newgrp
- /usr/bin/chfn
- /usr/bin/chsh
- /usr/bin/id

The BETA package has a *save* target in the Makefile, but it's commented out because different distributions place the programs in different places.

You should also make a backup copy of your `/etc/passwd` file, but be careful to name it something else if you place it in the same directory so you don't overwrite the `passwd` command.

4.4 Running make

You need to be logged as root to do most of the installation.

Run make to compile the executables in the package:

```
make all
```

You may see the warning: `rcsid defined but not used`. This is fine, it just happens because the author is using a version control package.

5.Installing

5.1 Have a boot disk handy in case you break anything.

If something goes terribly wrong, it would be handy to have a boot disk. If you have a boot/root combination from your installation, that will work, otherwise see the Bootdisk–HOWTO, which describes how to make a bootable disk.

5.2 Removing duplicate man pages

You should also move the manual pages that are about to be replaced. Even if you are brave enough install the Shadow Suite without making backups, you will still want to remove the old manual pages. The new manual pages won't normally overwrite the old ones because the old ones are probably compressed.

You can use a combination of: `man -aW command` and `locate command` to locate the manual pages that need to be (re)moved. It's generally easier to figure out which are the older pages before you run `make install`.

If you are using the Slackware 3.0 distribution, then the manual pages you want to remove are:

- /usr/man/man1/chfn.1.gz
- /usr/man/man1/chsh.1.gz
- /usr/man/man1/id.1.gz
- /usr/man/man1/login.1.gz
- /usr/man/man1/passwd.1.gz
- /usr/man/man1/su.1.gz
- /usr/man/man5/passwd.5.gz

There may also be man pages of the same name in the `/var/man/cat[1-9]` subdirectories that should also be deleted.

5.3 Running make install

You are now ready to type: (do this as root)

```
make install
```

This will install the new and replacement programs and fix−up the file permissions. It will also install the man pages.

This also takes care of installing the Shadow Suite include files in the correct places in `/usr/include/shadow`.

Using the BETA package you must manually copy the file `login.defs` to the `/etc` subdirectory and make sure that only *root* can make changes to it.

```
cp login.defs /etc
chmod 700 /etc/login.defs
```

This file is the configuration file for the *login* program. You should review and make changes to this file for your particular system. This is where you decide which tty's root can login from, and set other security policy settings (like password expiration defaults).

5.4 Running pwconv

The next step is to run `pwconv`. This must also be done as *root*, and is best done from the `/etc` subdirectory:

```
cd /etc
/usr/sbin/pwconv
```

`pwconv` takes your `/etc/passwd` file and strips out the fields to create two files: `/etc/npasswd` and `/etc/nshadow`.

A `pwunconv` program is also provided if you need to make a normal `/etc/passwd` file out of an `/etc/passwd` and `/etc/shadow` combination.

5.5 Renaming npasswd and nshadow

Now that you have run `pwconv` you have created the files `/etc/npasswd` and `/etc/nshadow`. These need to be copied over to `/etc/passwd` and `/etc/shadow`. We also want to make a backup copy of the original `/etc/passwd` file, and make sure only root can read it. We'll put the backup in root's home directory:

```
cd /etc
cp passwd ~passwd
chmod 600 ~passwd
mv npasswd passwd
mv nshadow shadow
```

You should also ensure that the file ownerships and permissions are correct. If you are going to be using *X−Windows*, the `xlock` and `xdm` programs need to be able to read the `shadow` file (but not write it).

There are two ways that this can be done. You can set `xlock` to suid root (`xdm` is usually run as root anyway). Or you can make the `shadow` file owned by `root` with a group of `shadow`, but before you do this, make sure that you have a shadow group (look in `/etc/group`). None of the users on the system should actually be in the shadow group.

```
chown root.root passwd
chown root.shadow shadow
chmod 0644 passwd
chmod 0640 shadow
```

Your system now has the password file shadowed. You *should* now pop over to another virtual terminal and verify that you can login.

Really, do this now!

If you can't, then something is wrong! To get back to a non−shadowed state, do the following the following:

```
cd /etc
cp ~passwd passwd
chmod 644 passwd
```

You would then restore the files that you saved earlier to their proper locations.

6.Other programs you may need to upgrade or patch

Even though the shadow suite contains replacement programs for most programs that need to access passwords, there are a few additional programs on most systems that require access to passwords.

If you are running a *Debian Distribution* (or even if you are not), you can obtain Debian sources for the programs that need to be rebuild from: ftp://ftp.debian.org/debian/stable/source/

The remainder of this section discusses how to upgrade `adduser`, `wu_ftpd`, `ftpd`, `pop3d`, `xlock`, `xdm` and `sudo` so that they support the shadow suite.

See the section Adding Shadow Support to a C program for a discussion on how to put shadow support into any other program that needs it (although the program must then be run SUID root or SGID shadow to be able to actually access the shadow file).

6.1 Slackware adduser program

Slackware distributions (and possibly some others) contain a interactive program for adding users called `/sbin/adduser`. A shadow version of this program can be obtained from ftp://sunsite.unc.edu/pub/Linux/system/Admin/accounts/adduser.shadow−1.4.tar.gz.

I would encourage you to use the programs that are supplied with the *Shadow Suite* (`useradd`, `usermod`, and `userdel`) instead of the slackware `adduser` program. They take a little time to learn how to use, but it's well worth the effort because you have much more control and they perform proper file locking on the `/etc/passwd` and `/etc/shadow` file (`adduser` doesn't).

See the section on Putting the Shadow Suite to use for more information.

But if you gotta have it, here is what you do:

```
tar −xzvf adduser.shadow−1.4.tar.gz
cd adduser
make clean
make adduser
chmod 700 adduser
```

```
cp adduser /sbin
```

6.2 The wu_ftpd Server

Most Linux systems some with the `wu_ftpd` server. If your distribution does not come with shadow installed, then your `wu_ftpd` will not be compiled for shadow. `wu_ftpd` is launched from `inetd/tcpd` as a *root* process. If you are running an old `wu_ftpd` daemon, you will want to upgrade it anyway because older ones had a bug that would allow the *root* account to be compromised (For more info see the Linux security home page).

Fortunately, you only need to get the source code and recompile it with shadow enabled.

If you are not running an ELF system, The `wu_ftp` server can be found on Sunsite as wu–ftp–2.4–fixed.tar.gz

Once you retrieve the server, put it in `/usr/src`, then type:

```
cd /usr/src
tar -xzvf wu-ftpd-2.4-fixed.tar.gz
cd wu-ftpd-2.4-fixed
cp ./src/config/config.lnx.shadow ./src/config/config.lnx
```

Then edit `./src/makefiles/Makefile.lnx`, and change the line:

```
LIBES    = -lbsd -support
```

to:

```
LIBES    = -lbsd -support -lshadow
```

Now you are ready to run the build script and install:

```
cd /usr/src/wu-ftpd-2.4-fixed
/usr/src/wu-ftp-2.4.fixed/build lnx
cp /usr/sbin/wu.ftpd /usr/sbin/wu.ftpd.old
cp ./bin/ftpd /usr/sbin/wu.ftpd
```

This uses the Linux shadow configuration file, compiles and installs the server.

On my Slackware 2.3 system I also had to do the following before running `build`:

```
cd /usr/include/netinet
ln -s in_systm.h in_system.h
```

```
cd -
```

Problems have been reported compiling this package under ELF systems, but the Beta version of the next release works fine. It can be found as wu−ftp−2.4.2−beta−10.tar.gz

Once you retrieve the server, put it in `/usr/src`, then type:

```
cd /usr/src
tar -xzvf wu-ftpd-2.4.2-beta-9.tar.gz
cd wu-ftpd-beta-9
cd ./src/config
```

Then edit `config.lnx`, and change:

```
#undef SHADOW.PASSWORD
```

to:

```
#define SHADOW.PASSWORD
```

Then,

```
cd ../Makefiles
```

and edit the file `Makefile.lnx` and change:

```
LIBES = -lsupport -lbsd # -lshadow
```

to:

```
LIBES = -lsupport -lbsd -lshadow
```

Then build and install:

```
cd ..
build lnx
cp /usr/sbin/wu.ftpd /usr/sbin/wu.ftpd.old
cp ./bin/ftpd /usr/sbin/wu.ftpd
```

Note that you should check your `/etc/inetd.conf` file to make sure that this is where your wu.ftpd server really lives. It has been reported that some distributions place the server daemons in different places, and then wu.ftpd in particular may be named something else.

6.3 Standard ftpd

If you are running the standard `ftpd` server, I would recommend that you upgrade to the `wu_ftpd` server. Aside from the known bug discussed above, it's generally thought to be more secure.

If you insist on the standard one, or you need *NIS* support, Sunsite has ftpd−shadow−nis.tgz

6.4 pop3d (Post Office Protocol 3)

If you need to support the third *Post Office Protocol (POP3)*, you will need to recompile a `pop3d` program. `pop3d` is normally run by `inetd/tcpd` as `root`.

There are two versions available from Sunsite: pop3d−1.00.4.linux.shadow.tar.gz and pop3d+shadow+elf.tar.gz

Both of these are fairly straight forward to install.

6.5 xlock

If you install the shadow suite, and then run *X Windows System* and lock the screen without upgrading your `xlock`, you will have to use `CNTL−ALT−Fx` to switch to another *tty*, login, and kill the `xlock` process (or use `CNTL−ALT−BS` to kill the X server). Fortunately it's fairly easy to upgrade your `xlock` program.

If you are running XFree86 Versions 3.x.x, you are probably using `xlockmore` (which is a great screen−saver in addition to a lock). This package supports *shadow* with a recompile. If you have an older `xlock`, I recommend that you upgrade to this one.

`xlockmore−3.5.tgz` is available at:
ftp://sunsite.unc.edu/pub/Linux/X11/xutils/screensavers/xlockmore−3.7.tgz

Basically, this is what you need to do:

Get the `xlockmore−3.7.tgz` file and put it in `/usr/src` unpack it:

```
tar −xzvf xlockmore−3.7.tgz
```

Edit the file: `/usr/X11R6/lib/X11/config/linux.cf`, and change the line:

```
#define HasShadowPasswd    NO

to

#define HasShadowPasswd    YES
```

Then build the executables:

```
cd /usr/src/xlockmore
xmkmf
make depend
make
```

Then move everything into place and update file ownerships and permissions:

```
cp xlock /usr/X11R6/bin/
cp XLock /var/X11R6/lib/app-defaults/
chown root.shadow /usr/X11R6/bin/xlock
chmod 2755 /usr/X11R6/bin/xlock
chown root.shadow /etc/shadow
chmod 640 /etc/shadow
```

Your xlock will now work correctly.

6.6 xdm

`xdm` is a program that presents a login screen for X–Windows. Some systems start `xdm` when the system is told to goto a specified run level (see `/etc/inittab`.

With the *Shadow Suite* install, `xdm` will need to be updated. Fortunately it's fairly easy to upgrade your `xdm` program.

`xdm.tar.gz` is available at: ftp://sunsite.unc.edu/pub/Linux/X11/xutils/xdm.tar.gz

Get the `xdm.tar.gz` file and put it in `/usr/src`, then to unpack it:

```
tar -xzvf xdm.tar.gz
```

Edit the file: `/usr/X11R6/lib/X11/config/linux.cf`, and change the line:

```
#define HasShadowPasswd    NO

to

#define HasShadowPasswd    YES
```

Then build the executables:

```
cd /usr/src/xdm
```

```
xmkmf
make depend
make
```

Then move everything into place:

```
cp xdm /usr/X11R6/bin/
```

`xdm` is run as *root* so you don't need to change it file permissions.

6.7 sudo

The program `sudo` allows a system administrator to let users run programs that would normally require root access. This is handy because it lets the administrator limit access to the root account itself while still allowing users to do things like mounting drives.

`sudo` needs to read passwords because it verifies the users password when it's invoked. `sudo` already runs SUID root, so accessing the `/etc/shadow` file is not a problem.

`sudo` for the shadow suite, is available as at:
ftp://sunsite.unc.edu/pub/Linux/system/Admin/sudo−1.2−shadow.tgz

Warning: When you install `sudo` your `/etc/sudoers` file will be replaced with a default one, so you need to make a backup of it if you have added anything to the default one. (you could also edit the Makefile and remove the line that copies the default file to `/etc`).

The package is already setup for shadow, so all that's required is to recompile the package (put it in `/usr/src`):

```
cd /usr/src
tar -xzvf sudo-1.2-shadow.tgz
cd sudo-1.2-shadow
make all
make install
```

6.9 pppd (Point−to−Point Protocol Server)

The pppd server can be setup to use several types of authentication: *Password Authentication Protocol* (PAP) and *Cryptographic Handshake Authentication Protocol* (CHAP). The pppd server usually reads the password strings that it uses from `/etc/ppp/chap-secrets` and/or `/etc/ppp/pap-secrets`. If you are using this default behavior of pppd, it is not necessary to reinstall pppd.

pppd also allows you to use the *login* parameter (either on the command line, or in the configuration or `options` file). If the *login* option is given, then pppd will use the `/etc/passwd` file for the username and passwords for the *PAP*. This, of course, will no longer work now that our password file is shadowed. For pppd−1.2.1d this requires adding code for shadow support.

The example given in the next section is adding shadow support to `pppd−1.2.1d` (an older version of pppd).

`pppd−2.2.0` already contains shadow support.

7.Putting the Shadow Suite to use.

This section discusses some of the things that you will want to know now that you have the *Shadow Suite* installed on your system. More information is contained in the manual pages for each command.

7.1 Adding, Modifying, and deleting users

The *Shadow Suite* added the following command line oriented commands for adding, modifying, and deleting users. You may also have installed the `adduser` program.

useradd

The `useradd` command can be used to add users to the system. You also invoke this command to change the default settings.

The first thing that you should do is to examine the default settings and make changes specific to your system:

```
useradd −D
```

```
GROUP=1
HOME=/home
INACTIVE=0
EXPIRE=0
SHELL=
SKEL=/etc/skel
```

The defaults are probably not what you want, so if you started adding users now you would have to specify all the information for each user. However, we can and should change the default values.

On my system:

- I want the default group to be 100
- I want passwords to expire every 60 days

* I don't want to lock an account because the password is expired
* I want to default shell to be `/bin/bash`

To make these changes I would use:

```
useradd -D -g100 -e60 -f0 -s/bin/bash
```

Now running `useradd -D` will give:

```
GROUP=100
HOME=/home
INACTIVE=0
EXPIRE=60
SHELL=/bin/bash
SKEL=/etc/skel
```

Just in case you wanted to know, these defaults are stored in the file `/etc/default/useradd`.

Now you can use `useradd` to add users to the system. For example, to add the user `fred`, using the defaults, you would use the following:

```
useradd -m -c "Fred Flintstone" fred
```

This will create the following entry in the `/etc/passwd` file:

```
fred:*:505:100:Fred Flintstone:/home/fred:/bin/bash
```

And the following entry in the `/etc/shadow` file:

```
fred:!:0:0:60:0:0:0:0
```

`fred`'s home directory will be created and the contents of `/etc/skel` will be copied there because of the `-m` switch.

Also, since we did not specify a UID, the next available one was used.

`fred`'s account is created, but `fred` still won't be able to login until we unlock the account. We do this by changing the password.

```
passwd fred
```

```
Changing password for fred
Enter the new password (minimum of 5 characters)
Please use a combination of upper and lower case letters and numbers.
New Password: *******
Re-enter new password: *******
```

Now the `/etc/shadow` will contain:

```
fred:J0C.WDR1amIt6:9559:0:60:0:0:0:0
```

And `fred` will now be able to login and use the system. The nice thing about `useradd` and the other programs that come with the *Shadow Suite* is that they make changes to the `/etc/passwd` and `/etc/shadow` files atomically. So if you are adding a user, and another user is changing their password at the same time, both operations will be performed correctly.

You should use the supplied commands rather than directly editing `/etc/passwd` and `/etc/shadow`. If you were editing the `/etc/shadow` file, and a user were to change his password while you are editing, and then you were to save the file you were editing, the user's password change would be lost.

Here is a small interactive script that adds users using `useradd` and `passwd`:

```
#!/bin/bash
#
# /sbin/newuser - A script to add users to the system using the Shadow
#                 Suite's useradd and passwd commands.
#
# Written my Mike Jackson <mhjack@tscnet.com> as an example for the Linux
# Shadow Password Howto.  Permission to use and modify is expressly granted.
#
# This could be modified to show the defaults and allow modification similar
# to the Slackware Adduser program.  It could also be modified to disallow
# stupid entries.  (i.e. better error checking).
#
##
#  Defaults for the useradd command
##
GROUP=100        # Default Group
HOME=/home       # Home directory location (/home/username)
SKEL=/etc/skel   # Skeleton Directory
INACTIVE=0       # Days after password expires to disable account (0=never)
EXPIRE=60        # Days that a passwords lasts
SHELL=/bin/bash  # Default Shell (full path)
##
#  Defaults for the passwd command
##
PASSMIN=0        # Days between password changes
PASSWARN=14      # Days before password expires that a warning is given
##
#  Ensure that root is running the script.
##
WHOAMI=`/usr/bin/whoami`
if [ $WHOAMI != "root" ]; then
        echo "You must be root to add news users!"
        exit 1
fi
##
#  Ask for username and fullname.
##
echo ""
echo -n "Username: "
read USERNAME
echo -n "Full name: "
read FULLNAME
#
```

```
echo "Adding user: $USERNAME."
#
# Note that the "" around $FULLNAME is required because this field is
# almost always going to contain at least on space, and without the "'s
# the useradd command would think that you we moving on to the next
# parameter when it reached the SPACE character.
#
/usr/sbin/useradd -c"$FULLNAME" -d$HOME/$USERNAME -e$EXPIRE \
        -f$INACTIVE -g$GROUP -m -k$SKEL -s$SHELL $USERNAME
##
#  Set password defaults
##
/bin/passwd -n $PASSMIN -w $PASSWARN $USERNAME >/dev/null 2>&1
##
#  Let the passwd command actually ask for password (twice)
##
/bin/passwd $USERNAME
##
#  Show what was done.
##
echo ""
echo "Entry from /etc/passwd:"
echo -n "    "
grep "$USERNAME:" /etc/passwd
echo "Entry from /etc/shadow:"
echo -n "    "
grep "$USERNAME:" /etc/shadow
echo "Summary output of the passwd command:"
echo -n "    "
passwd -S $USERNAME
echo ""
```

Using a script to add new users is really much more preferable than editing the `/etc/passwd` or `/etc/shadow` files directly or using a program like the Slackware `adduser` program. Feel free to use and modify this script for your particular system.

For more information on the `useradd` see the online manual page.

usermod

The `usermod` program is used to modify the information on a user. The switches are similar to the `useradd` program.

Let's say that you want to change `fred`'s shell, you would do the following:

```
usermod -s /bin/tcsh fred
```

Now `fred`'s `/etc/passwd` file entry would be change to this:

```
fred:*:505:100:Fred Flintstone:/home/fred:/bin/tcsh
```

Let's make `fred`'s account expire on 09/15/97:

```
usermod -e 09/15/97 fred
```

Now `fred`'s entry in `/etc/shadow` becomes:

```
fred:J0C.WDR1amIt6:9559:0:60:0:0:10119:0
```

For more information on the `usermod` command see the online manual page.

userdel

`userdel` does just what you would expect, it deletes the user's account. You simply use:

```
userdel -r username
```

The `-r` causes all files in the user's home directory to be removed along with the home directory itself. Files located in other file system will have to be searched for and deleted manually.

If you want to simply lock the account rather than delete it, use the `passwd` command instead.

7.2 The passwd command and passwd aging.

The `passwd` command has the obvious use of changing passwords. Additionally, it is used by the *root* user to:

- Lock and unlock accounts (`-l` and `-u`)
- Set the maximum number of days that a password remains valid (`-x`)
- Set the minimum days between password changes (`-n`)
- Sets the number of days of warning that a password is about to expire (`-w`)
- Sets the number of days after the password expires before the account is locked (`-i`)
- Allow viewing of account information in a clearer format (`-S`)

For example, let look again at `fred`

```
passwd -S fred
fred P 03/04/96 0 60 0 0
```

This means that `fred`'s password is valid, it was last changed on 03/04/96, it can be changed at any time, it expires after 60 days, fred will not be warned, and and the account won't be disabled when the password expires.

This simply means that if `fred` logs in after the password expires, he will be prompted for a new password at login.

If we decide that we want to warn `fred` 14 days before his password expires and make his account inactive 14 days after he lets it expire, we would need to do the following:

```
passwd -w14 -i14 fred
```

Now `fred` is changed to:

```
fred P 03/04/96 0 60 14 14
```

For more information on the `passwd` command see the online manual page.

7.3 The login.defs file.

The file `/etc/login` is the configuration file for the `login` program and also for the *Shadow Suite* as a whole.

`/etc/login` contains settings from what the prompts will look like to what the default expiration will be when a user changes his password.

The `/etc/login.defs` file is quite well documented just by the comments that are contained within it. However, there are a few things to note:

- It contains flags that can be turned on or off that determine the amount of logging that takes place.
- It contains pointers to other configuration files.
- It contains defaults assignments for things like password aging.

From the above list you can see that this is a rather important file, and you should make sure that it is present, and that the settings are what you desire for your system.

7.4 Group passwords.

The `/etc/groups` file may contain passwords that permit a user to become a member of a particular group. This function is enabled if you define the constant `SHADOWGRP` in the `/usr/src/shadow-YYMMDD/config.h` file.

If you define this constant and then compile, you must create an `/etc/gshadow` file to hold the group passwords and the group administrator information.

When you created the `/etc/shadow`, you used a program called `pwconv`, there no equivalent program to create the `/etc/gshadow` file, but it really doesn't matter, it takes care of itself.

To create the initial `/etc/gshadow` file do the following:

```
touch /etc/gshadow
chown root.root /etc/gshadow
chmod 700 /etc/gshadow
```

Once you create new groups, they will be added to the `/etc/group` and the `/etc/gshadow` files. If you modify a group by adding or removing users or changing the group password, the `/etc/gshadow` file will be changed.

The programs `groups`, `groupadd`, `groupmod`, and `groupdel` are provided as part of the *Shadow Suite* to modify groups.

The format of the `/etc/group` file is as follows:

```
groupname:!:GID:member,member,...
```

Where:

groupname

The name of the group

!

The field that normally holds the password, but that is now relocated to the `/etc/gshadow` file.

GID

The numerical group ID number

member

List of group members

The format of the `/etc/gshadow` file is as follows:

```
groupname:password:admin,admin,...:member,member,...
```

Where:

groupname

The name of the group

password

The encoded group password.

admin

List of group administrators

member

List of group members

The command `gpasswd` is used only for adding or removing administrators and members to or from a group. `root` or someone in the list of administrators may add or remove group members.

The groups password can be changed using the `passwd` command by *root* or anyone listed as an administrator for the group.

Despite the fact that there is not currently a manual page for `gpasswd`, typing `gpasswd` without any parameters gives a listing of options. It's fairly easy to grasp how it all works once you understand the file formats and the concepts.

7.5 Consistency checking programs

pwck

The program `pwck` is provided to provide a consistency check on the `/etc/passwd` and `/etc/shadow` files. It will check each username and verify that it has the following:

- the correct number of fields
- unique user name
- valid user and group identifier
- valid primary group
- valid home directory
- valid login shell

It will also warn of any account that has no password.

It's a good idea to run `pwck` after installing the *Shadow Suite*. It's also a good idea to run it periodically, perhaps weekly or monthly. If you use the `-r` option, you can use `cron` to run it on a regular basis and have the report mailed to you.

grpck

`grpck` is the consistency checking program for the `/etc/group` and `/etc/gshadow` files. It performs the following checks:

- the correct number of fields
- unique group name
- valid list of members and administrators

It also has the `-r` option for automated reports.

7.6 Dial−up passwords.

Dial−up passwords are another optional line of defense for systems that allow dial−in access. If you have a system that allows many people to connect locally or via a network, but you want to limit who can dial in and connect, then dial−up passwords are for you. To enable dial−up passwords, you must edit the file `/etc/login.defs` and ensure that `DIALUPS_CHECK_ENAB` is set to `yes`.

Two files contain the dial−up information, `/etc/dialups` which contains the ttys (one per line, with the leading "/dev/" removed). If a tty is listed then dial−up checks are performed.

The second file is the `/etc/d_passwd` file. This file contains the fully qualified path name of a shell, followed by an optional password.

If a user logs into a line that is listed in `/etc/dialups`, and his shell is listed in the file `/etc/d_passwd` he will be allowed access only by suppling the correct password.

Another useful purpose for using dial−up passwords might be to setup a line that only allows a certain type of connect (perhaps a PPP or UUCP connection). If a user tries to get another type of connection (i.e. a list of shells), he must know a password to use the line.

Before you can use the dial−up feature, you must create the files.

The command `dpasswd` is provided to assign passwords to the shells in the `/etc/d_passwd` file. See the manual page for more information.

8.Adding shadow support to a C program

Adding shadow support to a program is actually fairly straightforward. The only problem is that the program must be run by root (or SUID root) in order for the the program to be able to access the `/etc/shadow` file.

This presents one big problem: very careful programming practices must be followed when creating SUID programs. For instance, if a program has a shell escape, this must not occur as root if the program is SUID root.

For adding shadow support to a program so that it can check passwords, but otherwise does need to run as root, it's a lot safer to run the program SUID shadow instead. The `xlock` program is an example of this.

In the example given below, `pppd−1.2.1d` already runs SUID as root, so adding shadow support should not make the program any more vulnerable.

8.1 Header files

The header files should reside in /usr/include/shadow. There should also be a /usr/include/shadow.h, but it will be a symbolic link to /usr/include/shadow/shadow.h.

To add shadow support to a program, you need to include the header files:

```
#include <shadow/shadow.h>
#include <shadow/pwauth.h>
```

It might be a good idea to use compiler directives to conditionally compile the shadow code (I do in the example below).

8.2 libshadow.a library

When you installed the *Shadow Suite* the libshadow.a file was created and installed in /usr/lib.

When compiling shadow support into a program, the linker needs to be told to include the libshadow.a library into the link.

This is done by:

```
gcc program.c -o program -lshadow
```

However, as we will see in the example below, most large programs use a Makefile, and usually have a variable called LIBS=... that we will modify.

8.3 Shadow Structure

The libshadow.a library uses a structure called spwd for the information it retrieves from the /etc/shadow file. This is the definition of the spwd structure from the /usr/include/shadow/shadow.h header file:

```
struct spwd
{
  char *sp_namp;               /* login name */
  char *sp_pwdp;               /* encrypted password */
  sptime sp_lstchg;            /* date of last change */
  sptime sp_min;               /* minimum number of days between changes */
  sptime sp_max;               /* maximum number of days between changes */
  sptime sp_warn;              /* number of days of warning before password
                                  expires */
  sptime sp_inact;             /* number of days after password expires
                                  until the account becomes unusable. */
  sptime sp_expire;            /* days since 1/1/70 until account expires
*/
  unsigned long sp_flag;       /* reserved for future use */
```

```
};
```

The *Shadow Suite* can put things into the `sp_pwdp` field besides just the encoded passwd. The password field could contain:

```
username:Npge08pfz4wuk;@/sbin/extra:9479:0:10000::::
```

This means that in addition to the password, the program `/sbin/extra` should be called for further authentication. The program called will get passed the username and a switch that indicates why it's being called. See the file `/usr/include/shadow/pwauth.h` and the source code for `pwauth.c` for more information.

What this means is that we should use the function `pwauth` to perform the actual authentication, as it will take care of the secondary authentication as well. The example below does this.

The author of the *Shadow Suite* indicates that since most programs in existence don't do this, and that it may be removed or changed in future versions of the *Shadow Suite*.

8.4 Shadow Functions

The `shadow.h` file also contains the function prototypes for the functions contained in the `libshadow.a` library:

```
extern void setspent __P ((void));
extern void endspent __P ((void));
extern struct spwd *sgetspent __P ((__const char *__string));
extern struct spwd *fgetspent __P ((FILE *__fp));
extern struct spwd *getspent __P ((void));
extern struct spwd *getspnam __P ((__const char *__name));
extern int putspent __P ((__const struct spwd *__sp, FILE *__fp));
```

The function that we are going to use in the example is: `getspnam` which will retrieve for us a `spwd` structure for the supplied name.

8.5 Example

This is an example of adding shadow support to a program that needs it, but does not have it by default.

This example uses the *Point−to−Point Protocol Server* (pppd−1.2.1d), which has a mode in which it performs *PAP* authentication using user names and passwords from the `/etc/passwd` file instead of the *PAP* or *CHAP* files. You would not need to add this code to `pppd−2.2.0` because it's already there.

This feature of pppd probably isn't used very much, but if you installed the *Shadow Suite*, it won't work anymore because the passwords are no longer stored in `/etc/passwd`.

The code for authenticating users under `pppd-1.2.1d` is located in the `/usr/src/pppd-1.2.1d/pppd/auth.c` file.

The following code needs to be added to the top of the file where all the other `#include` directives are. We have surrounded the `#includes` with conditional directives (i.e. only include if we are compiling for shadow support).

```
#ifdef HAS_SHADOW
#include <shadow.h>
#include <shadow/pwauth.h>
#endif
```

The next thing to do is to modify the actual code. We are still making changes to the `auth.c` file.

Function `auth.c` before modifications:

```
/*
 * login - Check the user name and password against the system
 * password database, and login the user if OK.
 *
 * returns:
 *      UPAP_AUTHNAK: Login failed.
 *      UPAP_AUTHACK: Login succeeded.
 * In either case, msg points to an appropriate message.
 */
static int
login(user, passwd, msg, msglen)
    char *user;
    char *passwd;
    char **msg;
    int *msglen;
{
    struct passwd *pw;
    char *epasswd;
    char *tty;

    if ((pw = getpwnam(user)) == NULL) {
        return (UPAP_AUTHNAK);
    }
     /*
     * XXX If no passwd, let them login without one.
     */
    if (pw->pw_passwd == '\0') {
        return (UPAP_AUTHACK);
    }

    epasswd = crypt(passwd, pw->pw_passwd);
    if (strcmp(epasswd, pw->pw_passwd)) {
        return (UPAP_AUTHNAK);
    }

    syslog(LOG_INFO, "user %s logged in", user);

    /*
     * Write a wtmp entry for this user.
     */
    tty = strrchr(devname, '/');
    if (tty == NULL)
```

```
        tty = devname;
    else
        tty++;
    logwtmp(tty, user, "");             /* Add wtmp login entry */
    logged_in = TRUE;

    return (UPAP_AUTHACK);
}
```

The user's password is placed into `pw->pw_passwd`, so all we really need to do is add the function `getspnam`. This will put the password into `spwd->sp_pwdp`.

We will add the function `pwauth` to perform the actual authentication. This will automatically perform secondary authentication if the shadow file is setup for it.

Function `auth.c` after modifications to support shadow:

```
/*
 * login - Check the user name and password against the system
 * password database, and login the user if OK.
 *
 * This function has been modified to support the Linux Shadow Password
 * Suite if USE_SHADOW is defined.
 *
 * returns:
 *      UPAP_AUTHNAK: Login failed.
 *      UPAP_AUTHACK: Login succeeded.
 * In either case, msg points to an appropriate message.
 */
static int
login(user, passwd, msg, msglen)
    char *user;
    char *passwd;
    char **msg;
    int *msglen;
{
    struct passwd *pw;
    char *epasswd;
    char *tty;

#ifdef USE_SHADOW
    struct spwd *spwd;
    struct spwd *getspnam();
#endif

    if ((pw = getpwnam(user)) == NULL) {
        return (UPAP_AUTHNAK);
    }

#ifdef USE_SHADOW
        spwd = getspnam(user);
        if (spwd)
                pw->pw_passwd = spwd->sp-pwdp;
#endif

     /*
      * XXX If no passwd, let NOT them login without one.
      */
    if (pw->pw_passwd == '\0') {
```

```
        return (UPAP_AUTHNAK);
    }
#ifdef HAS_SHADOW
    if ((pw->pw_passwd && pw->pw_passwd[0] == '@'
         && pw_auth (pw->pw_passwd+1, pw->pw_name, PW_LOGIN, NULL))
        || !valid (passwd, pw)) {
        return (UPAP_AUTHNAK);
    }
#else
    epasswd = crypt(passwd, pw->pw_passwd);
    if (strcmp(epasswd, pw->pw_passwd)) {
        return (UPAP_AUTHNAK);
    }
#endif

    syslog(LOG_INFO, "user %s logged in", user);

    /*
     * Write a wtmp entry for this user.
     */
    tty = strrchr(devname, '/');
    if (tty == NULL)
        tty = devname;
    else
        tty++;
    logwtmp(tty, user, "");             /* Add wtmp login entry */
    logged_in = TRUE;

    return (UPAP_AUTHACK);
}
```

Careful examination will reveal that we made another change as well. The original version allowed access (returned `UPAP_AUTHACK` if there was NO password in the `/etc/passwd` file. This is *not* good, because a common use of this login feature is to use one account to allow access to the PPP process and then check the username and password supplied by PAP with the username in the `/etc/passwd` file and the password in the `/etc/shadow` file.

So if we had set the original version up to run as the shell for a user i.e. `ppp`, then anyone could get a ppp connection by setting their PAP to user `ppp` and a password of null.

We fixed this also by returning `UPAP_AUTHNAK` instead of `UPAP_AUTHACK` if the password field was empty.

Interestingly enough, `pppd-2.2.0` has the same problem.

Next we need to modify the Makefile so that two things occur: `USE_SHADOW` must be defined, and `libshadow.a` needs to be added to the linking process.

Edit the Makefile, and add:

```
LIBS = -lshadow
```

Then we find the line:

```
COMPILE_FLAGS = -I.. -D_linux_=1 -DGIDSET_TYPE=gid_t
```

And change it to:

```
COMPILE_FLAGS = -I.. -D_linux_=1 -DGIDSET_TYPE=gid_t -DUSE_SHADOW
```

Now make and install.

9.Frequently Asked Questions.

Q: I used to control which tty's *root* could log into using the file `/etc/securettys`, but it doesn't seem to work anymore, what's going on?

A: The file `/etc/securettys` does absolutely nothing now that the *Shadow Suite* is installed. The tty's that *root* can use are now located in the login configuration file `/etc/login.defs`. The entry in this file may point to another file.

Q: I installed the *Shadow Suite*, but now I can't login, what did I miss?

A: You probably installed the Shadow programs, but didn't run `pwconv` or you forgot to copy `/etc/npasswd` to `/etc/passwd` and `/etc/nshadow` to `/etc/shadow`. Also, you may need to copy `login.defs` to `/etc`.

Q: In the section on xlock, it said to change the group ownership of the `/etc/shadow` file to `shadow`. I don't have a `shadow` group, what do I do?

A: You can add one. Simply edit the `/etc/group` file, and insert a line for the shadow group. You need to ensure that the group number is not used by another group, and you need to insert it before the `nogroup` entry. Or you can simply suid `xlock` to root.

Q: Is there a mailing list for the Linux Shadow Password Suite?

A: Yes, but it's for the development and beta testing of the next Shadow Suite for Linux. You can get added to the list by mailing to: `shadow-list-request@neptune.cin.net` with a subject of: `subscribe`. The list is actually for discussions of the Linux `shadow-YYMMSS` series of releases. You should join if you want to get involved in further development or if you install the Suite on your system and want to get information on newer releases.

Q: I installed the *Shadow Suite*, but when I use the `userdel` command, I get "userdel: cannot open shadow group file", what did I do wrong?

A: You compiled the *Shadow Suite* with the `SHADOWGRP` option enabled, but you don't have an `/etc/gshadow` file. You need to either edit the `config.h` file and recompile, or create an `/etc/group` file. See the section on shadow groups.

Q: I installed the *Shadow Suite* but now I'm getting encoded passwords back in my `/etc/passwd` file, what's wrong?

A: You either enabled the `AUTOSHADOW` option in the Shadow `config.h` file, or your `libc` was compiled with the `SAHDOW_COMPAT` option. You need to determine which is the problem, and recompile.

10.Copyright Message.

11.Miscellaneous and Acknowledgments.

The code examples for `auth.c` are taken from pppd−1.2.1d and ppp−2.1.0e, Copyright (c) 1993 and The Australian National University and Copyright (c) 1989 Carnegie Mellon University.

Thanks to Marek Michalkiewicz <marekm@i17linuxb.ists.pwr.wroc.pl> for writing and maintaining the *Shadow Suite* for Linux, and for his review and comments on this document.

Thanks to Ron Tidd <rtidd@tscnet.com> for his helpful review and testing.

Thanks to everyone who has sent me feedback to help improve this document.

Please, if you have any comments or suggestions then mail them to me.

regards

Michael H. Jackson <mhjack@tscnet.com>