

# Chapter 15

## System Security

This chapter introduces basic security issues of the Linux system. This provides the user with a foundation of the ability to prevent unwanted remote users access to their system. More advanced topics exist which the administrator must research.

### Concepts Learned in this Chapter

- Data Encryption
- Secure Access
- Tracking Data modification
- System user security

## Table of Contents

System Security.....	1
15.1 Password Security .....	4
15.2 Data Encryption of Files, .....	4
15.2.1 A Little Theory on Encryption.....	4
15.2.2 GPG Encryption .....	6
15.2.3 Creating a Symmetric Encryption File .....	9
15.2.4 Digital Signature.....	10
15.2.5 Other Options.....	11
15.2.6 These Commands Are Too Long .....	11
15.2.6 Examples of Commands.....	12
15.3 Secure Shell (Not Complete).....	13
15.3.1 Step by Step Procedure.....	13
15.3.2 Sneaker Net.....	15
15.3.3 Email Attachment .....	15
15.3.4 SSH Connection .....	15
15.3.5 Logging onto Remote Host .....	15
15.3.6 Enhancing your logon on your Remote Access .....	16
15.4 Secure Copy (scp) .....	16
15.5 Secure File Transfer (sftp) .....	16
15.6 Improved Sudo Security.....	16
15.7 OpenSSL.....	19
15.8 Tripwire (Initial Thoughts) .....	19
15.9 Secure TTY .....	19
15.10.1 Why Use PAM .....	21
15.10.2 Overview of How it Works .....	21
15.10.3 Configuration Files .....	21
15.10.4 Authentication Modules .....	22
15.10.5 Module Format.....	22
15.10.5.1 Module-Type .....	22
15.10.5.2 Control-Flag .....	22
15.10.5.3 Test-File .....	23
15.10.5.4 Arguments.....	23
15.10.6 Examples of the Test Files include.....	23
15.10.7 Discussion of Example.....	24
15.10.8 Testing for Conditions.....	25
15.11 Snort .....	26
15.12 SATAN and SAINT.....	26
15.13 Chroot.....	26
15.14 Port Map.....	26
15.15 Controlling Remote Access.....	26
15.15.1 Access Control.....	27
15.15.2 Access Control Rules.....	27
15.15.3 Rule Patterns.....	27
15.15.4 Wildcard Matches.....	28
15.15.5 Operators.....	28

15.15.6 Shell Commands.....	28
15.15.7 Server Endpoint Patterns.....	29
15.15.8 Detecting Address Spoofing Attacks.....	29
15.15.9 Examples.....	29
15.15.9.1 Mostly Closed.....	29
15.15.9.2 Mostly Open.....	29
15.15.10 Diagnostics.....	30
15.16 TCP Wrappers.....	30
15.17 Commands Used in this Chapter.....	30
15.18 Chapter Review Questions.....	30

## 15.1 Password Security

We have made the issue multiple times previously, but it must be emphasized that a user's password is the first line of defense. Do not compromise your password.

## 15.2 Data Encryption of Files<sup>1,2</sup>

In order to insure privacy of information on the Internet, it may be a wise idea to encrypt the files – even the email that we wish to send across the Internet. Sometime we may find it necessary to encrypt the information that we maintain on our own system.

### 15.2.1 A Little Theory on Encryption

All through the ages, many various attempts have been made to maintain secrecy of information. Most were successful, and a few very important encryption techniques failed.

Encryption of analog information was a necessity in the early 1900's through World War II, but after the war we entered the digital age. The method of encryption of an analog signal was quite involved – but the encryption of digital information is really quite simple.

Our first requirement is to understand another type of **Boolean Algebra**. We create a special truth table to demonstrate what we need:

Input A	Input B	Output t
0	0	0
0	1	1
1	0	1
1	1	0

This type of table is called an Exclusive OR, where the only time you obtain a positive output is when you have one and only one valid input.

Now if we have some information (in digital format) such as:

1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 1 1 1

Now we make up a key, say one that is 10 bits long, such as:

1 1 1 0 1 1 0 1 0 0

We now perform an Exclusive OR between the information and the key, repeating the key as necessary.

1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 1 1 1	original data
1 1 1 0 1 1 0 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 0 1 1 0 1 0 0	key x 3

<sup>1</sup> David Scribner, member North Texas Linux Users Group ([www.ntlug.org](http://www.ntlug.org)), [www.bigfoot.com/~dscribner](http://www.bigfoot.com/~dscribner) (a special thanks for helping me learn)

<sup>2</sup> gnupg.org: GPG Privacy Handbook and GPG Mini HowTo

And get:

0 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1      encrypted data

This is the data string that is actually transmitted across the network. At the receiving end we need to decrypt the received bits in order to understand the original data. We do this by again performing an Exclusive OR on the received data, thus:

0 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1      received data  
1 1 1 0 1 1 0 1 0 0 **1 1 1 0 1 1 0 1 0 0** 1 1 1 0 1 1 0 1 0 0      key x 3

And get:

1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 1 1 1      decrypted data

Comparing to the original data:

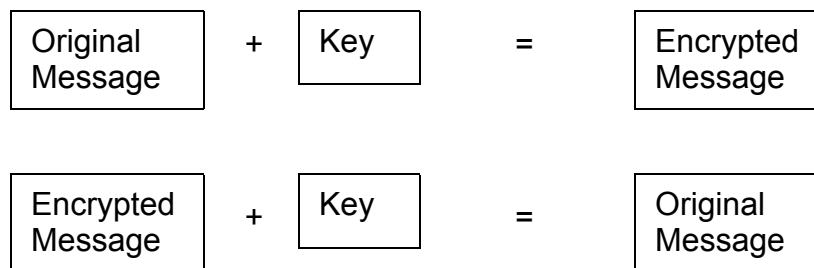
1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 1 1 1      original data

We see that we now have our original information back.

Now this is a very simple case. Our key is only 10 bits long. What if we made the key 56, 64, 128, or 1024 or even 2048 bits long? Then we would have a key that would be much more difficult to figure out.

Other algorithms exist, such as the “Caesar’s Cipher” and “ROT13”, but these are beyond the scope of this discussion.

Now comes the case where we need to transmit the key to the party that needs to decrypt our message. Lets say User A wants to transmit an encrypted message to User B. User A must have the encryption key in order to encrypt the message. User B must also have the key in order to decrypt the message. This leads to a situation where there are at least two places where the key may be stolen.



In order for User B to decrypt the message, he must have the key. He must either be given the key prior to going on his mission, or have it transmitted to him, which might subject the key to being found out by the party that we are attempting to keep the information from. This form of encryption is referred to as **symmetric** or conventional encryption.

Another technique in encryption is to establish a double set of keys. The first one is called a **public key** and is made available to anyone who wishes to send an encrypted message to you. When you create a public key, you also create a **secret key**, and this is used to decrypt the publicly encrypted message. Thus one does not have to publish their secret key. Only the user holding the secret key is able to decrypt the message. The secret key is also referred to as the “Private Key”.

Today, there are two major encryption tools for encrypting messages. Both of them use the double key method. The difference is that the first, “**Pretty Good Privacy (PGP)**” is restricted due to legal issues. The second is “**GNU Privacy Guard (GPG)**” and is open source and available for public use – and it is FREE. (For some this is the only justification.) GPG uses the same encryption algorithm as PGP, but does not use the same software code in order to not violate any license. Philip Zimmerman originated the PGP algorithm and later made it as an open standard, thus allowing others to develop the process (specifically GNUPG.org).

PGP includes several additional features that GPG does not. GUNPG has been developed under the Unix philosophy of “doing one thing, and doing it well”. This means that if one desires additional features (‘bell and whistles’), you will have to utilize other front ends, such as GPA (GNU Privacy Assistant), which perform other features, leaving the encryption process to GPG.

The rest of this lab will focus on the GPG encryption process.

### 15.2.2 GPG Encryption

GPG comes with Linux as a default installation. One only needs to run it in order to make it active. To activate, follow the following process:

1. Issue the command:  
**gpg --gen-key**
2. This creates the \$HOME / .gnupg directory (first time only) and exits.
3. Again issue the command:  
**gpg --gen-key**
4. The file **secring.gpg** is created, this is the secret decryption key (after answering the following questions).
5. The file **pubring.gpg** is created, this is the public encryption key (after answering the following questions).
6. Select key type:
  1. DSA and ElGamal (default)
  2. DSA (sign only)
  3. ElGamal (sign and encrypt)
 > select the default value normally
7. Specify key size:
  1. 1024 bits (default)
  2. minimum 768 bits
  3. maximum 2048 bits
8. Expiration date:
  1. Does not expire      0
  2. Days                      n
  3. Weeks                      nw
  4. Months                   nm
  5. Years                    ny
 > for a starter, and learning, we should select 0

9. Supply the user information:
  1. Real Name
  2. Email Address
  3. Comment (optional)
10. Supply a Passphrase:  
This is a sentence that you can remember and that no one else can guess. After you type it in, it will generally not be long enough, so you will have to add some more random key strokes / mouse movements to complete the full key. For this discussion, lets make the passphrase "Linux GPG encryption".
11. The above two keys are now completed and are stored in the .gnupg directory. Every user on a system can generate their own key set, which is stored in their respective home directory.

If you issue the command:

**gpg --list-keys**

You will observe something like:

```
pub 1024D / 420D7X5Z 2002-08-15 usera (prof) prof@school.edu
sub 1024g / X2m1234
```

where

pub	public key
sub	secret key
1024	encryption key size value (number of bits)
D	DSA encryption
g	ElGamal signing key
420D7X5Z	key ID
2002.8.15	date key created (Aug 15, 2002)
usera	user's name
(prof)	user's comment
<u>prof@school.edu</u>	user's email address

Now you want to have your associate send you a private message, you need to send him your public key that he then uses to encrypt the message. Your associate will then transmit the encrypted message to you, which you will then be able to decrypt using your secret key.

When you have created your public key, you commonly put it on your web server so that anyone who wishes to send you an encrypted message may download it. After they have your public key, they add it to their public keyring. This keyring may contain just a few, or several hundred public encryption keys – one for each individual that you wishes to send a message to.

There are a large number of options available when using gpg. The following sets an example of how one might proceed with the encryption and decryption of a message.

User A, needs to receive an encrypted file.

Creates public and secret keys.

**gpg --gen-key**

Now we need to export the key into an ASCII format:

```
gpg --export --armor --output opkey.asc
```

This creates an ASCII file (armor option) of the key that may be transmitted across the Internet with no problems, and names it “opkey.asc”. It is now published on the web site.

This would export all of your public keys. If we have only our own, it would be acceptable, but we normally only desire to export our own, so we issue the alternative command:

```
gpg --export 420D7X5Z --armor --output opkey
```

The key ID identifies which public key is to be extracted. You may also identify the key by using the user name or email address.

User B, retrieves User A public key.

Issues the command to merge the UserA key to their public keyring.

```
gpg --import opkey.asc
```

Verifies that the key is available on the public keyring by issuing the command:

```
gpg --list-keys
```

Where we will see that one of the keys is the one from UserA. The output will be something like:

```
pub 1024D/309E5B2C 2002-09-22 John Doe (student)  
<jdoe@isp.com  
sub 1024g/D9F59876 2002-09-22
```

```
pub 1024D/420D7X5Z 2002-08-15 usera (prof)  
<prof@school.edu>  
sub 1024g/B7C94729 2002-09-22
```

User B now encrypts the file using UserA’s public key, issuing the command:

```
gpg --encrypt --recipient usera {encrypted-file-name}      or  
gpg -e -r usera {encrypted-file-name}  
e.g.      gpg -e -r prof jobstat
```

This produces an encrypted ASCII text file called **{encrypted-file-name}.gpg** (jobstat.gpg) that can be transmitted over the Internet using usera’s (prof) public key.

User B now attaches the encrypted message to an email to User A.

User A receives the encrypted message from User B. To read the file, issue the command:

```
gpg --decrypt ofile.gpg  
e.g.      gpg --decrypt jobstat
```



Our output is:

***You need a passphrase to unlock the secret key for user “usera (prof) prof@school.edu” 1024 bit ELG-E key, ID EF7473B, created 2002-08-15 (main key ID 420D7X5Z)***

***Enter passphrase: Linux GPG encryption*** (not visible)

We now have the decrypted file efile on our system screen, and we can read it. If we wish to save the file, use the command:

```
gpg --decrypt ofile.gpg --output dfile.txt  
e.g.          gpg --decrypt jobstat.gpg --output jobstatus.txt
```

Now we want to add a little more security to our file than just the encryption. We want to digitally sign it in order to prove that we actually transmitted it. This is kind of like adding a Cyclic Redundancy Check (CRC) to the file – but more powerful.

As the sender, we can add our signature to the encryption by issuing the command:

```
gpg --encrypt --sign --recipient usera --armor ofile  
Enter passphrase: Userb passphrase
```

This adds the originator’s digital signature to the encrypted file. This is not additional encryption, but it does add additional data that insures that we are the originator of the document. On decryption, we will get the following:

***You need a passphrase to unlock the secret key for user “usera (prof) prof@school.edu” 1024 bit ELG-E key, ID EF7473B, created 2002-08-15 (main key ID 420D7X5Z)***

***Enter passphrase: Linux GPG encryption*** (not visible)

***gpg: Signature made Mon 23 Sep 2002 2:05 PM CDT using DSA key ID 0C409498***

***gpg: Good signature from John Doe (student),  
jdoe@isp.com>***

### 15.2.3 Creating a Symmetric Encryption File

Sometimes you might desire to encrypt sensitive data on your own computer, such as a file that you keep your passwords in. You do not intend to send it to anyone, but it sure is hard remembering all of those different passwords. To create a symmetric encryption, issue the command:

```
gpg --symmetric filename
```

This will generate the file ‘filename.gpg’. For additional security, you might want to make the passphrase different from that used for the asymmetric passphrase. In fact this is a good idea.

### 15.2.4 Digital Signature

It is often convenient to Digitally Sign a document, rather than fully encrypt it. There are two options for creating a digitally signed document, compressed and clear signed.

Digital signing works in a reverse manner to normal encryption. Whereas the originator of a document wants to encrypt his document by using your public key and you decrypt it using your private key, when using digital signature, you the originator use your private key and the recipient uses your public key. Note that the recipient can only verify that the signature is valid, since the original message is not encrypted.

Lets set up an example of creating a message that is to be signed. For our example we will use a very simple text document:

simple.txt

Now is the time for all students to work hard.

Since we previously created our public and private keys, they will be used to digitally sign the document. This procedure is as follows:

```
$ gpg --output simple.sig --sign simple.txt
```

*You need a passphrase to unlock the secret key for  
user: "Dennis Rice (da Prof) <drice@dal.devry.edu>"  
1024-bit DSA key, ID 677FD40F, created 2004-06-02*

*Enter passphrase: xxxx your-passphrase xxxx*

The outputted file is the following:

```
$ cat simple.sig
£}WqfnAN^a^IEÃ¾Ä÷~ùå
Ä
%©
%¹©
iùE
99
```

The file is not encrypted – but it is compressed.

If we want to output a file in a clear text mode, then we use a slightly different command:

```
$ gpg --clearsign simple.txt
```

*You need a passphrase to unlock the secret key for  
user: "Dennis Rice (da Prof) <drice@dal.devry.edu>"  
1024-bit DSA key, ID 677FD40F, created 2004-06-02*

*Enter passphrase: xxxx your-passphrase xxxx*

The outputted file is the following:

```
$ cat simple.txt.asc
```

**-----BEGIN PGP SIGNED MESSAGE-----**

**Hash: SHA1**

***Now is the time for all students to work hard.***

**-----BEGIN PGP SIGNATURE-----**

**Version: GnuPG v1.0.6 (GNU/Linux)**

**Comment: For info see <http://www.gnupg.org>**

***iD8DBQFAvIOpLRGJjmd/1A8RAh8pAKCE1N+w6KyDTRxh97kHhZtcD  
PRzvQCeLMb/***

***WhEms45Xp/YQ4rt6jZXd1pg=  
=1Dlj***

**-----END PGP SIGNATURE-----**

The above message with the signature is the one that is transmitted over the Internet. The file is normally sent as an attachment.

Finally, we need to verify that the received message is the one that was sent, and has not been altered. The recipient would have the originator's public key, so they would issue the following command:

```
$ gpg --output simple.text --decrypt simple.txt.asc  
gpg: Signature made Wed 02 Jun 2004 05:24:41 PM CDT using DSA  
key ID 677FD40F  
gpg: Good signature from "Dennis Rice (da Prof)  
<drice@dal.devry.edu>"
```

The message has been transmitted and verified that it has not been tampered with.

If the file had been tampered with, then the following output would be observed:

```
$ gpg --output simple.text2 --decrypt simple.txt.asc  
gpg: Signature made Wed 02 Jun 2004 05:24:41 PM CDT using DSA  
key ID 677FD40F  
gpg: BAD signature from "Dennis Rice (da Prof)  
<drice@dal.devry.edu>"
```

So we can observe that the message received was bad and should not be considered valid.

### 15.2.5 Other Options

Naturally many additional options exist, but these must be left to the reader to research. For reference, two sites are recommended for additional reading:

**[www.bigfoot.com/~dscribner](http://www.bigfoot.com/~dscribner)** □ download for reference documents.

**[www.gnupg.org](http://www.gnupg.org)** □ gnu Privacy Guard site - Documentation.

### 15.2.6 These Commands Are Too Long

Ok, it is a real pain to type in all of those commands. Well, some may be abbreviated. The following is a list that may be used in an abbreviated format:

<u>Full Command</u>	<u>Short Command</u>
--sign	-s
--detach-sign	-b
--encrypt	-e
--symmetric	-c
--recipient	-r
--armor	-a
--output	-o
--verbose	-v
--decrypt	-d

### 15.2.6 Examples of Commands

All of the above provides a lot of options – providing you with more opportunity to make an error!

Export an ASCII key:

```
gpg -a --export 'A User' > profkey.asc
```

This command exports the key for 'A User' (placed in quotes because there is a space in the name) producing an ASCII file called profkey. Instead of using the -o option to create a file, the director has been used.

Import an ASCII key:

```
gpg --import profkey.asc
```

This command takes the ASCII key file 'profkey.asc' and adds it to the users public keyring.

Encrypt a file using another user's public key:

```
gpg -ea -r 'B User' file2encrypt
```

This command creates a new ASCII file (a option) by the name of file2encrypt.gpg using the public code of 'B User'.

Decrypt a file using your secret key, which was encrypted with your public key:

```
gpg -d file2decrypt.gpg
```

You must be logged in as the user that is to decrypt the message, and you are asked for the users Passphrase. The output will be to the screen. To create a new file, either use the -o option or use the director to create a new file.

Create a symmetric encrypted file:

```
gpg -c file2encrypt
```

You will be required to enter your Passphrase twice to encrypt the file. Since the passphrase may be different from your "regular" passphrase – entering the passphrase twice insures there are no typos. If this were not the case, you might never be able to decrypt it!

Decrypt a symmetric encrypted file:

```
gpg -d file2decrypt
```

You will be required to enter your Passphrase in order to decrypt. Output is to the screen, if you want it to a file, either use the `-o` option or the director.

### **15.3 Secure Shell (Not Complete)**

As was noted in a previous chapter, when one establishes a connection to a remote system using Telnet, all information sent across the network is transmitted in clear text. This includes your username and password – not a good thing! Hence, anyone monitoring the network will be able to see what you are doing on the remote system. If you were configuring the system as the administrator, they would then know how to access it at a later date and to then make modifications unknown to you. Even worse, they could create a new user with root privileges.

To improve security, a protocol was developed to encrypt all information except for the IP header information. Hence anyone monitoring your transmissions will find encrypted data and will not be able to detect what you are doing. This protocol is called **Secure Shell**, or **ssh**. Previously in Chapter 8, Remote Administration, we discussed the basic usage of a Secure Shell. Here we provide additional information on how to enhance the level of security.

Secure Shell comes in two levels – basic and enhanced. Basic ssh is enabled by just establishing a link between yourself and the remote server using a default encryption. Although the information is encrypted, it is not as secure as one might desire – in fact it may be breakable by someone who wants to monitor your transmission. Enhanced ssh utilizes a public / private key process so that you may improve the security level by using your public / private keys.

There are two versions of ssh presently utilized. Version 2 supports version 1 and improves security, so we will only discuss this one option.

As a basic service, when one logs onto a remote system and the enhance security has not been enabled, you will receive a normal password prompt. All information will be encrypted.

The normal installation of ssh on Red Hat is maintained in the `/usr/bin` directory. Since this directory is normally accessible to all users, each user may create their own keys, which will be maintained in their `$HOME` directory.

At this time, there are three encryption formats that ssh supports – `rsa1` (one), `dsa`, and `rsa`. Thus we must create 3 different sets of keys if we need to connect to different systems. As a general rule, the `dsa` encryption is often taken as the default. In the process of creating the three formats, we need to generate two for each – public and private.

Finally, we need to edit our `sshd_config` file to limit support of ssh for version 2 only. It is maintained in the `/etc/ssh` directory.

#### **15.3.1 Step by Step Procedure**

It is assumed at this point that the server ssh daemon has already been installed and is operational. The following is the setup of the enhanced secure service. For Red Hat, this is a default configuration, but the user may not have

implemented the service by creating a public ring. Note that the ssh encryption is not the same as that generated by GPG.

1. First we need to log onto the server system (the one we will log onto from a remote system) as the administrator. We then issue the following commands:

```
ssh-keygen -b 1024 -t rsa1  
ssh-keygen -b 1024 -t dsa  
ssh-keygen -b 1024 -t rsa
```

In general, there is no need to use the rsa1 key unless you know a system is limited to version 1.

You will be requested for the file location (take the default), and for a passphrase. After you enter your phrase, DO NOT FORGET IT!

Edit the /etc/ssh/sshd\_config file. Change the line:

```
#Protocol 1,2          to  
Protocol 2
```

This will insure that we are using only ssh version 2.

2. Log out as the administrator and back in as your normal username. Again create the sets of files. Only create the keys for dsa and rsa, unless you also wish to support version 1, then also create the key for rsa1.

```
ssh-keygen -b 1024 -t dsa  
ssh-keygen -b 1024 -t rsa  
ssh-keygen -b 1024 -t rsa1          (only if desired)
```

You will again be requested for the file location and passphrase.

Now to explain what we have done. ssh-keygen is the program that generates the public and private keys; -b specifies the bit length of the encryption key; and -t specifies the type of key to be used. In our example we have specified a 1024 bit key for the dsa, rsa, and rsa version 1 encryption algorithms. Finally, when we modified the sshd\_config file, we specified that we would only accept the version 2 protocol. Since we modified the configuration file, we will need to restart the daemon. Issue the command:

```
service sshd restart
```

Process 1 and 2 both generate a different set of keys, each in the respective \$HOME directory.

3. Now you need to transmit your public key to the remote user. We could do this the sneaker-foot way (giving the user a floppy disk with your public key), or we could send it directly to him (or her). Once again we have options – you could send it just as an attachment and have the other user append it to their

authorized\_keys2 file, or you could send and attach it yourself, of course with a “few minor complications”.

### 15.3.2 Sneaker Net

For the easy method, copy your user public key to a floppy:

```
cd                this changes to your home directory
cd .ssh           changes to the hidden directory .ssh
mcopy id_dsa.pub a: copies to a DOS formatted floppy
```

Now give your floppy to your associate and have them append it to their authorized\_keys2 file:

```
cd
cd .ssh
mcopy a:id_dsa.pub
cat id_dsa.pub >> authorized_keys2
```

Your public code is now appended to their authorization key file. Make sure you use the double >, otherwise their authorized\_keys2 will be replaced totally with your key, and they will have lost all previous keys. (Might this be a good area to recommend backup of the file before appending?) Note that you hand the floppy disk directly to the recipient, not to someone that will hand it off to the person it is intended for – that could be a breach in your security.

### 15.3.3 Email Attachment

If you send the file as an attachment to an email, the other user must then save the file and append it as above. Note that there is some security risk to this if someone should intercept your email.

### 15.3.4 SSH Connection

In this case, you can send the public file directly to the remote user's file and attach it directly without their intervention. Of course, they must have implemented their key so you can attach yours to it. The requirement here is that you must be supplied an appropriate password to gain access to their system. To do this, issue the following command:

```
ssh username@remote-host "cat >> .ssh/authorized_keys2" <
id_dsa.pub
```

Before you can append your key, you will have to provide the user password, hence you must be a user on the remote system. Note that in using this process, you will be transmitting the file across the network in an encrypted mode. Much better!

### 15.3.5 Logging onto Remote Host

OK, now you have transferred your public file to the remote system, and you need to log on.

To log on using your username (the one you transferred the public key to), issue the command:

**ssh username@remote-host**

Now you will be requested for the passphrase rather than your password. We have enhanced remote access security.

### **15.3.6 Enhancing your logon on your Remote Access**

OK, you have improved security, but heck, you have to always type in that passphrase. You don't mind the phrase, just hate all those typing errors ☐.

It would not be sufficient to have a process to just pass the passphrase – we want to make sure that it is you that is logging on. During an automatic login, when we ssh to access the remote system, a challenge is made of the passphrase and a response of the private key is used to verify the challenge.

To enable this process, we need to utilize two additional ssh-utilities.

TBC

### **15.4 Secure Copy (scp)**

Secure Copy, which was introduced back in Chapter 5, allows for the transfer of a file between two hosts in a secure format. **scp** uses the same encryption technique as ssh. To set up scp, first set up the ssh encryption.

### **15.5 Secure File Transfer (sftp)**

Secure File Transfer was also introduced back in Chapter 5. This performs exactly the same as the normal ftp, except that the transfer of data is encrypted. Again, it uses the encryption process created by ssh, so that must be established first.

### **15.6 Improved Sudo Security**

Sudo was first covered back in Chapter 3, showing how one could log in as a normal user and then upgrade to the administrator on a command basis. For example, if the user were part of the group wheel, then the following command could be issued:

**\$ sudo cat /etc/passwd**

As the administrator, you may wish to limit what someone is allowed to perform. This can be accomplished by setting up specific rights for different users. This information is maintained in the `/etc/sudoers` file. Editing of this file should be implemented using the `visudo` command.

The `/etc/sudoers` file is used to validate users and what commands they may issue. Additionally, sudo can log both successful and unsuccessful login attempts, as authentication is verified through PAM (covered shortly).

The sudoers file consists of two type of entries, aliases and user specifications. Aliases are variables, specifications state who is allowed to what.



**Aliases** follow the four following formats:

```
User_Alias:      NAME = User_List
Runas_Alias:     NAME = Runas_List
Host_Alias:      NAME = Host_List
Cmnd_Alias:      NAME = Cmnd_List
```

The “NAME” is always in all-capitals. The list that follows is comma delimited and may span more than one line by using the continuation character (\).

Setting up the sudoers file is a complex process, the on-line manual must be consulted for more detail (man sudo and sudoers). The easiest way to show how it works is with examples.

```
User_Alias:
User_Alias FULLTIMERS = drice, jdoe, msmith
User_Alias PARTTIMERS = wking, iward, wkrause
User_Alias OFFICERS = jwhite, bkertner
User_Alias WEBMASTERS = ehoffer, tstaley
```

Here we have set up four different groups of users, assigning various users to each group.

```
Runas_Alias:
Runas_Alias OP = root, operator
Runas_Alias DB = mysql, postgresql
```

```
Host_Alias:
Host_Alias SPARC = bigtime, eclipse
Host_Alias PC = dlug, router1, wkstn1, wkstn2
Host_Alias SERVERS = bigtime, dlug, www
```

Here we have classified different types of host architecture, and assigned the hostname of each accordingly.

```
Cmnd_Alias:
Cmnd_Alias DUMPS = /usr/bin/dump, /usr/sbin/restore
Cmnd_Alias KILL = /usr/bin/kill
Cmnd_Alias SHUTDOWN = /usr/sbin/shutdown
Cmnd_Alias HALT = /usr/sbin/halt, /usr/sbin/fasthalt
Cmnd_Alias SU = /usr/bin/su
```

In our final example, we are a generic name for various commands. That way we can specify a set of commands to a specific User\_Alias without having to specify each command.

User Specifications is the section that designates who is allowed to issue specific commands. The format of the command follows the syntax:

where:

**Who** specifies a specific user, designated users (User\_Alias), or a group of users.

**Source** specifies what hosts may issue commands.

**Permissions** specifies authentication requirements.

**Commands** specifies what commands may be issued.

By default, two entities are already set up, the user root and the group %wheel.

```
root          ALL = (ALL)  ALL
%wheel        ALL = (ALL)  ALL
```

This specifies that both root and anyone a member of the group wheel (denoted by the “%”) may run any command from any host as any user.

We can assign specific rights to specific users, or the group of users specified by the `User_Alias`.

FULLTIMERS ALL = NOPASSWD: ALL

In this line, all users of the FULLTIMES group may run any command from any host, and do not require authentication.

Assignment of a different group for different rights might be set like the following.

PARTTIMERS ALL = ALL

Here, the part-time administrators may run any command from any host, but they are required to authenticate with a password.

A specific user may be allowed to issue commands from a specific set of hosts.

wilbur                      PC = ALL

In this example, the user wilbur is allowed to run all commands from only the PC hosts.

A set up users may be allowed to issue specific commands.

OFFICERS ALL = DUMPS, KILL, SHUTDOWN, HALT

Now a user that is a member of the OFFICERS group is allowed to run only those commands that are specified by DUMPS, KILL, SHUTDOWN, or HALT.

We can limit to which username that a user may su to.

```
johnny      ALL = /usr/bin/su WEBMASTERS
```

In this case, the user johnny can use the su command, logging in as one of the WEBMASTERS users. In this case, johnny must know the username and password for the user that he will log in as.

There are requirements to allow a specific group to change the password for anyone else – but we do not want to allow the root administrator's password to be modified.

```
PARTTIMERS    SPARC = /usr/bin/passwd [A-z]*, !/usr/bin/passwd
root
```

Now all users assigned to the PARTTIMERS group may modify the passwords for any other user, except root, as long as the first letter of the password starts with an alpha character, either upper or lower case. Additionally, they may only issue the changes from one of the SPARC hosts.

A restriction for who is allowed to work on the web page may be set up.

```
WEBMASTERS    www = (www) ALL, (root) /usr/bin/su www
```

The users that are part of the WEBMASTERS group, when on the www host, may su onto the host as the user www.

The above represents a lot of examples. More are available in the man page (sudoers).

When setting up users, it is best to create different groups, and then assign different users to the group. Users change often, but groups don't. Changing a group's membership is far easier than changing the sudoers file.

## **15.7        OpenSSL**

### **15.8        Tripwire (Initial Thoughts)**

Topic outline

1. Run program /etc/tripwire/twinstall.sh
2. Run program tripwire --init
3. Create a site passphrase
4. Create a local passphrase
5. To complete the installation, need to enter the site and local passphrases previously created
6. Must enter local passphrase to create an initial database of files on your system

Files created:

- |    |          |                                |
|----|----------|--------------------------------|
| 1. | hostname | local passphrase encrypted key |
| 2. | sitekey  | site passphrase encrypted key  |
| 3. | tw.cfg   | tripwire configuration         |
| 4. | tw.pol   | tripwire policy file           |

### **15.9        Secure TTY**

It is one issue to log onto a system as the root administrator from the system keyboard, it is another issue when a user telnets into the system and logs in as the root administrator. It would be a great advantage if we could limit who a telnet user could log in as.

Unix and Linux provide a mechanism to do this just this. The administrator can specify specifically who if telnet user is allowed to log in as "root" or not. The means to this is the file **/etc/securetty**.

```

auth      required    /lib/security/pam_securetty.so
auth      required    /lib/security/pam_stack.so
service=system-auth
auth      required    /lib/security/pam_nologin.so
account   required    /lib/security/pam_stack.so
service=system-auth
password  required    /lib/security/pam_stack.so
service=system-auth
session   required    /lib/security/pam_stack.so
service=system-auth
session   optional    /lib/security/pam_console.so

```

This file is a mapping of virtual consoles (vc) to remote access terminals (tty). This process applies to both remote access and to the ALT (F1 – F6) alternate terminal keys. At first the format of the file is not obvious, but there is a direct correlation between each vc and tty entry. The format of the file is:

```

vc/1
vc/2
vc/3
...
tty1
tty2
tty3
...

```

The file is a single column and has been shortened for brevity.

Of special note –

**tty1 is the normal keyboard / monitor, and must not be restricted.**

To prevent a user from logging in as the root administrator, simply place an “#” in front of the specific tty that is to be limited, such as:

```
#tty4
```

Now a user can log into tty as any user except root. Once logged in, the user may switch user (su) or “sudo”.

### **15.10 Pluggable Authentication Modules (PAM)**<sup>3,4</sup>

Pam was originally developed for the Solaris Unix Operating System by Sun Microsystems, and has been enhanced by the Linux community.

PAM is a method that Unix and Linux utilize to verify the rights of users to perform various functions in accordance to system authentication policies. The configuration files for specific applications reside in the **/etc/pam.d** directory.

The concept of PAM is for a PAM aware application to test whether a user has rights to operate it. Our first method of security is just the password, but PAM adds additional features that allow the testing of individual users, testing

<sup>3</sup> Official Red Hat Linux Administration's Guide, Red Hat Press, Wiley Publishing

<sup>4</sup> Red Hat Linux Security and Optimization, Mohammad Kabir, Red Hat Press, Wiley Publishing

their point of origin and even the time of day. Many “Security Objects” (.so files) are available to test various conditions.

So what is a “PAM aware application? When an application is created, the software developer must encode special features into it to allow testing of the user via the PAM modules.

When an application is PAM aware, a file by the same application name must exist within the directory **/etc/pam.d**. The application then opens this PAM file to test the rights of the user. If the user passes the desired tests, then the program continues to function, otherwise it is terminated.

The PAM file contains a number of module lines, each of which tests a given feature of the user. In some cases, if the user passes the conditions of the feature, then the next test is made, and in others, if the user fails the conditions, the system immediately kills the application’s operation. Of course, there are options that when a user fails the conditions, additional testing is continued, but the security test still fails and the application is killed; the advantage of this is that the user is unable to know for which reason the test failed.

### 15.10.1 Why Use PAM

PAM provides a number of advantages when used properly.

1. Common authentication mythology for applications.
2. Great flexibility and control over applications
3. Allows developers to utilize an authentication standard rather than developing their own

### 15.10.2 Overview of How it Works

Lets say you write a PAM aware program – that is one that must verify the user prior to continuing to run.

In order to continue, various user attributes are checked and verified by comparing the username, password, access rights, time of day, and other attributes that one may deem necessary.

When the program starts, in this example, the user must login and submit a password. The program then runs a specific configuration file in the **/etc/pam.d/** directory. This configuration file specifies a file of authentication modules that are used to test the username, password and other desired attributes. The scripts that are used to test for authentication are located in the **/lib/security** directory.

### 15.10.3 Configuration Files

Originally, there was a single file, **/etc/pam.conf** that was used for configuration. This file is now only used if the **/etc/pam.d/** directory does not exist.

A PAM configuration file is given the name of its service that it applies to, not necessarily that of the application. For example, for logging in, the common PAM application is **login**. When a user logs in, the file used for authentication is **/etc/pam.d/login**

```

]# cat login
#%PAM-1.0
auth    required  /lib/security/pam_securetty.so
auth    required  /lib/security/pam_stack.so service=system-auth
auth    required  /lib/security/pam_nologin.so
account required  /lib/security/pam_stack.so service=system-auth
password required /lib/security/pam_stack.so service=system-auth
session required  /lib/security/pam_stack.so service=system-auth
session optional  /lib/security/pam_console.so
]#

```

Below we will investigate the layout and options of this particular file.

### 15.10.4 Authentication Modules

Each line in the file specifies a different type of **authentication module**. The order of the modules in a given service file is important to the authentication process. Thus various tests may be made to insure that a user is properly authorized.

### 15.10.5 Module Format

Each module is normally set up as follows:

Module-Type	Control-Flag	Test-File	Arguments
-------------	--------------	-----------	-----------

#### 15.10.5.1 Module-Type

There are four types of authentication Modules-Types that are used to control access to a service.

- **auth** A module used to request a password and verify it. Group membership or Kerberos are utilized through this module.
- **account** Verifies that access is allowed, such as insuring that the account has not expired.
- **password** Used to set passwords
- **session** Used to manage a user's session after they have been authenticated. May be used to specify the user's home directory and mailbox.

#### 15.10.5.2 Control-Flag

There are four types of Control-Flag that are use to specify ....

- **required** The module must be successfully completed in order to provide authentication. Failure of a required module causes the user's access to fail, but will not be notified until after all other tests are completed.
- **requisite** The module must successfully be completed in order for user authentication. If the module fails, the user is notified immediately.

- **sufficient** If no required or requisite modules have previously failed, and the sufficient module passes, no further modules are checked. Thus the user is authenticated.
- **optional** Other module checks are ignored if this module fails.

Additional Control-Flags are available and are documented in the **/usr/share/doc/pam-version-number/** directory.

#### 15.10.5.3 Test-File

The Test-File is the path/file that the PAM module is to be tested against. All Test-Files are located in the **/lib/security/** directory. The files that actually perform the authentication testing are known as a “security object” files, and hence are tagged with a “.so” suffix.

#### 15.10.5.4 Arguments

Arguments may be used to pass additional information to the module. This allows the same authentication program to be used by different modules, thus testing for different requirements.

#### 15.10.6 Examples of the Test Files include

The following are examples of some of the authentication modules located in the **/lib/security/** directory:

- **pam\_access.so** Uses the **/etc/security/access.conf** configuration file. This is a list of usernames, which specify which tty-list or host-list is to be utilized.
  - tty-list** Restricts users from logging on via a console (tty) port.
  - host-list** Restricts individual users or domains from access.
- **pam\_console.so** Specifies which PAM-aware commands, such as shutdown, halt, and reboot an ordinary user may use. Called by the login program – login, gdm, or kdm.
- **pam\_cracklib.so** Checks if the password has expired (per pam\_unix.so), then the user is prompted for a new password. The new password is then checked for validity.
- **pam\_deny.so** This module always returns a failure. Used to insure that a login fails. This condition should be the last line in every authentication module to insure that if all other tests are indecisive, the general test will fail.
- **pam\_env.so** Sets the environmental values specified in the **/etc/security/pam\_env.conf** file.
- **pam\_ftp.so** Tests for ftp user’s email address, which is used for anonymous login password.
- **pam\_issue.so** Displays the contents of the **/etc/issue** file at the user login.
- **pam\_lastlog.so** Displays the user’s last login information.
- **pam\_limits.so** Limits resources of an ordinary user as specified in the **/etc/security/limits.conf** file.

- **pam\_listfile.so** Provides restrictions to remote user login.
- **pam\_localuser.so** Checks to see if a user has unread mail, and displays a message if true.
- **pam\_mail.so** Notifies user if they have new mail.
- **pam\_motd.so** Displays the message of the of the day when the user logs in.
- **pam\_nologin.so** Check to verify that the user is “root”. If not the module fails. Only the administrator is allowed to log into the system. The file **/etc/nologin** must exist for this to be effective, which contains a short banner specifying that the user, if not root, is not allowed to log in.
- **pam\_pwcheck.so** Used to verify the password. Works in conjunction with the **pam\_unix.so** test file.
- **pam\_permit.so** Used for very low-risk applications, which all users are allowed to perform.
- **pam\_radius.so** Authenticates RADIUS users.
- **pam\_rhosts.so** Specifies whether a user may log in on as a remote user.
- **pam\_securetty.so** Verifies that the root user is allowed to log in on a specified tty port, as specified in the **/etc/securetty** file.
- **pam\_tally.so** Tracks user login attempts, denying access if exceeded.
- **pam\_time.so** Verifies that a user may log on during specified times.
- **pam\_unix.so** Account verification is performed. Initiates the process for the user to be asked for a password. Checks against the **passwd** and **shadow** files to check if the account has expired (if the **shadow** argument exists). Allows the user to log in with a blank password (if the **nullok** argument exists) – not a good idea!
- **pam\_userdb.so** Authenticates various user databases.
- **Pam\_wheel.so** Allows only wheel users to log on.

### 15.10.7 Discussion of Example

Now lets go back and look at the previously shown example for ftp.

**auth required /lib/security/pam\_securetty.so**

**pam\_securetty** tests verifies that the administrator is allowed to log in on the respective port. The **/etc/securetty** file specifies which ports (ALT-Fx) ports the administrator is allowed to use. If the administrator is not allowed on the designated tty port, then the test fails.

**auth required /lib/security/pam\_stack.so service=system-auth**

**pam\_stack** tests the user’s password to verify that it is correct.

**auth required /lib/security/pam\_nologin.so**

**pam\_nologin** tests to see if the user is logging is as the administrator. If not, then the tests fails.



**account required /lib/security/pam\_stack.so service=system-auth**

pam\_stack again tests to verify that the account is still valid, that is the password has not expired or the account has not been disabled.

**password required /lib/security/pam\_stack.so service=system-auth**

pam\_stack again tests to see that when the user elects to modify their password that the password meets the minimum requirements.

**session required /lib/security/pam\_stack.so service=system-auth**

pam\_stack again is used to set up the user's home directory and mailbox.

**session optional /lib/security/pam\_console.so**

pam\_console specifies which administrative commands a user is allowed to issue, i.e., may the user halt the system.

### 15.10.8 Testing for Conditions

When developing a set of test conditions, which pam module should be used? The following provides examples of different situations.

<u>Objective</u>	<u>Module</u>	<u>Action</u>
Root Only	pam_nologin	Create /etc/nologin file with message
Root	pam_cracklib	Change /etc/system-auth pam_cracklib
Complexity	line to have the following various options:	
type=Linux minlen=7	password length 7 characters	
dcredit = 1	at least 1 number	
ucredit = 1	at least 1 upper case character	
lcredit = 1	at least 1 lower case character	
ocredit = 1	at least 1 symbol	
Password History	pam_pwcheck	Add "remember=3" to the pam_pwcheck line of /etc/pam.d/passwd. This forces the user to use three different passwords on password change. Old passwords are maintained in the /etc/security/opasswd file.
Limit	pam_login	Add line to the file:
Connections	session	required pam_limits.so Edit the /etc/security/limits.conf file by adding * hard maxlogins 2
Time of Day	pam_login	Add line to the file: account required pam_time.so

Add line to the **/etc/security/time.conf** file  
 \*; \*; timeout; username <hhmm>-<hhmm>  
 User is now limited to logging in during hours  
 other than those specified.

Password supply Aging	pam_unix	Use the application <b>chage</b> <b>username</b> and values to limit password aging.
Account supply Expiration	pam_unix	Use the application <b>chage</b> <b>username</b> and values to specify expiration.
Grace <b>username</b> and supply	pam_unix	Use the application <b>chage</b>
Logins		values to specify grace logins.
Log Activity  file.	pam_warn	Add to the <b>/etc/pam.d/login</b> file session optional pam_warn.so Logins will be logged to the /var/log/messages

### 15.11 Snort

### 15.12 SATAN and SAINT

### 15.13 Chroot

### 15.14 Port Map

### 15.15 Controlling Remote Access

Access to a system may be controlled by two files, **/etc/hosts.allow** and **/etc/hosts.deny**. Between these two, the administrator may allow or limit which hosts are allowed to gain access. Note that access is controlled by host name or IP address. Individual users, even if valid on a system, if logging in from a restricted host, would not be permitted on the system.

In this review, the basic options will be reviewed, additional detail may be obtained from the `hosts_options(5)` man page.

### 15.15.1 Access Control

Access control is specified by various entries into the two files. The first file to be evaluated when a remote host attempts a logon is the **hosts.allow** file, after that the **hosts.deny** will be evaluated. If the request does not meet or fail either limitations in the two files, access is granted.

### 15.15.2 Access Control Rules

The rules for access control is made up of zero or more rule lines. They are processed in order of appearance, and is terminated when a matching rule is found.

A newline character is ignored if the line is terminated in a backslash ( \ ). This permits a long line to span multiple lines of the screen.

Blank lines or line that begin with the hash ( # ) character are ignored. This allows comments to be written into the rules.

All remaining lines must follow the following format, with brackets specifying options.

```
daemon_list : client_list [ : shell_command ]
```

A `daemon_list` is a list of one or more daemon process names or a wildcard.

A `client_list` is a list of one or more host names, addresses, patterns or wildcards that are to be matched against the client host name or address.

The list elements must be separated by blanks and / or commas.

### 15.15.3 Rule Patterns

The Access Control List (ACL) utilizes a pattern for its language.

A string that begins with the dot ( . ) character is matched if the last components of its name match the specified pattern. For example, the pattern “`.tue.nl`” will find a match in the hostname of “`wzv.win.tue.nl`”.

A string that ends with the dot ( . ) character is matched if the host address if the first numeric field matches the given string. For example, the pattern “`123.135.`” matches the address of every host on the specified network (123.135.X.X).

A string that begins with the AT-Sign ( @ ) character is treated as an NIS netgroup name. A host name is matched if it is a host member of the specified netgroup.

A numeric IP Address in the form of “`A.B.C.D / M.N.P.R`” is interpreted a network / subnet mask pair for an IPv4 network host. All hosts on the specified network address are a match.

A numeric IP Address in the form of “A:B:C:D:E:F:G:H / M” (in Hex) is interpreted as an IPv6 network / subnet mask pair (mask in CIDR format). All hosts on the specified network address are a match.

A string that begins with a slash ( / ) character is treated as a file name. A host name or address is matched if it matches any host name or address pattern listed in the specified file name. The external file format is zero or more lines which lists a host name or address pattern separated by whitespace. This allows an external file to contain the list of hosts or addresses that are to be tested.

The asterisk ( \* ) or question mark ( ? ) characters are wild cards. An asterisk matches multiple characters whereas a question mark matches a single character. These substitutions may be made for either a hostname or IP Address, except for either of the IP “Address / Subnet Mask” format or with a hostname matching format that either begins or ends with the dot.

#### 15.15.4 Wildcard Matches

Explicit wildcard support the following:

<b>ALL</b>	The universal wildcard, always matches.
<b>LOCAL</b>	Matches any host whose name does not contain a dot character.
<b>UNKOWN</b>	Matches any user whose name is unknown. This pattern should be used with caution especially if using a remote server and it becomes unavailable.
<b>KNOWN</b>	Matches any known username that is known. This pattern should be used with caution especially if using a remote server and it becomes unavailable.
<b>PARANOID</b>	Matches any host whose name does not match its address. When the tcpd is built with “-DPARANOID (default mode), it drops request requests from the client even before looking at the Access Control tables.

#### 15.15.5 Operators

The following operator is available.

<b>EXCEPT</b>	Intended to exclude a portion of a list. The form of the operator is “ <b>List-1 EXCEPT List-2</b> ”. This is interpreted as matching anything within List-1 unless it is included in List-2.
---------------	---

#### 15.15.6 Shell Commands

Allows shell commands to be included. The command is subjected to substitutions. For more detail, see the man pages.

### 15.15.7 Server Endpoint Patterns

In order to distinguish clients by the network address, use the pattern:

**process\_name@host\_pattern : client\_list ...**

This format allows a host with different hostname from the specified network address to be matched. This condition might be used to impede a user from logging into a system and using a invalid network address.

### 15.15.8 Detecting Address Spoofing Attacks

Due to a flaw in some TCP/IP implementation, intruders may sometime be able to break into a system via a remote shell service. For more detail, see the man pages for hosts-allow and hosts-deny.

### 15.15.9 Examples

In reviewing the above one rules, one can observe that a great deal of flexibility exists to either allow or deny either an individual host or a network of hosts.

Note that the allow table is viewed first, then the deny table.

#### 15.15.9.1 Mostly Closed

In this example, access is denied to all by default with only explicitly authorized hosts permitted.

/etc/hosts.allow:

ALL: LOCAL @some\_netgroup

ALL: .foobar.edu EXCEPT terminalserver.foobar.edu

/etc/hosts.deny:

ALL: ALL

All users of the network some\_netgroup and users of the foobar.edu network, except the host terminal server are permitted. Everyone else is denied. Note that if we viewed the hosts.deny file first, no one would be allowed in – thus by viewing the allow file first, those who qualify may access, those who do not are denied.

#### 15.15.9.2 Mostly Open

By default, with nothing in the hosts.allow file, access is granted. To limit those which are to be denied, the following entries are required in the deny file.

/etc/hosts.deny:

ALL: some.host.name, .some.domain

ALL EXCEPT in.fingerd: other.host.name, .other.domain

The first rule denies the specific host “some.host.name”. The second rule permits the Internet finger request from the network “other.domain” and the host “other.host.name”. See the man page for hosts.allow for more detail.

**15.15.10 Diagnostics**

When permitting or denying access and a syntax error is incurred in the appropriate file, an entry is made via the syslog daemon.

**15.16 TCP Wrappers<sup>5</sup>**

TCP Wrappers provide a means to guard against remotely required services such as ssh, telnet, and ftp. The function of TCP Wrapper is to “wrap” a service request by authentication service, thus providing a greater control of access and logging. When utilized, TCP Wrapper provides the remote user host identification information.

By default, TCP Wrappers is installed in Red Hat 8 and later for server systems. Earlier versions require installation.

**15.17 Commands Used in this Chapter****15.18 Chapter Review Questions**

---

<sup>5</sup> Official Red Hat Linux Administrator's Guide, Red Hat Press, Wiley Publishing



Host_Alias	17	encrypt	8p.
Runas_Alias	17	export	8
User_Alias	17	gen-key	6
		import	8
TCP Wrappers	30	list-keys	7p.
Tripwire	19	sign	10
		service	
URL		sshd	14
GNUPG.org	6	ssh	
Utility		keygen	14
chage	26	sudo	16
gpg		visudo	16
armor	8		
clearsign	10		
decrypt	8, 11	www.bigfoot.com	11
		www.gnupg.org	11