



# Hardening Basics Part 2

1. Which SSH Protocol version is the most secure?

ans: **SSH2**

2. This is a random, arbitrary number, used as the session key, that is used to encrypt GPG?

ans: **nonce**

A nonce (which stands for "number used once") is a term commonly used in computer science, cryptography, and security. It refers to a random or pseudo-random number that is used only once in a specific context or transaction.

3. Yey/Ney - GPG is based off of the OpenPGP standard

ans: **yey**

4. What is the command to generate your GPG keys?

ans: **gpg -gen-key**

The `gpg --gen-key` command is a command-line instruction used to generate a GPG (GNU Privacy Guard) key pair. GPG is an open-source encryption and digital signature tool that provides cryptographic privacy and authentication for data communication

5. What is the command to symmetrically encrypt a file with GPG?

ans: **gpg -c**

The `gpg -c` command is used to encrypt a file using GPG (GNU Privacy Guard) with symmetric encryption.

6. **What is the command to asymmetrically encrypt a file with GPG?**

ans: **gpg -e**

7. **What is the command to create SSH keys?**

ans: **ssh-keygen**

8. **Where are ssh keys stored in a user's home directory?**

ans : **.ssh**

9. **What option needs to be set to select the type of key to generate for SSH?**

ans: **-t**

10. **The SSH configuration options presented in this chapter were found in what file (full path)?**

ans: **/etc/ssh/sshd\_config**

---

Type:- 3

## Encrypting Your Files

Encrypting Your Files with GPG

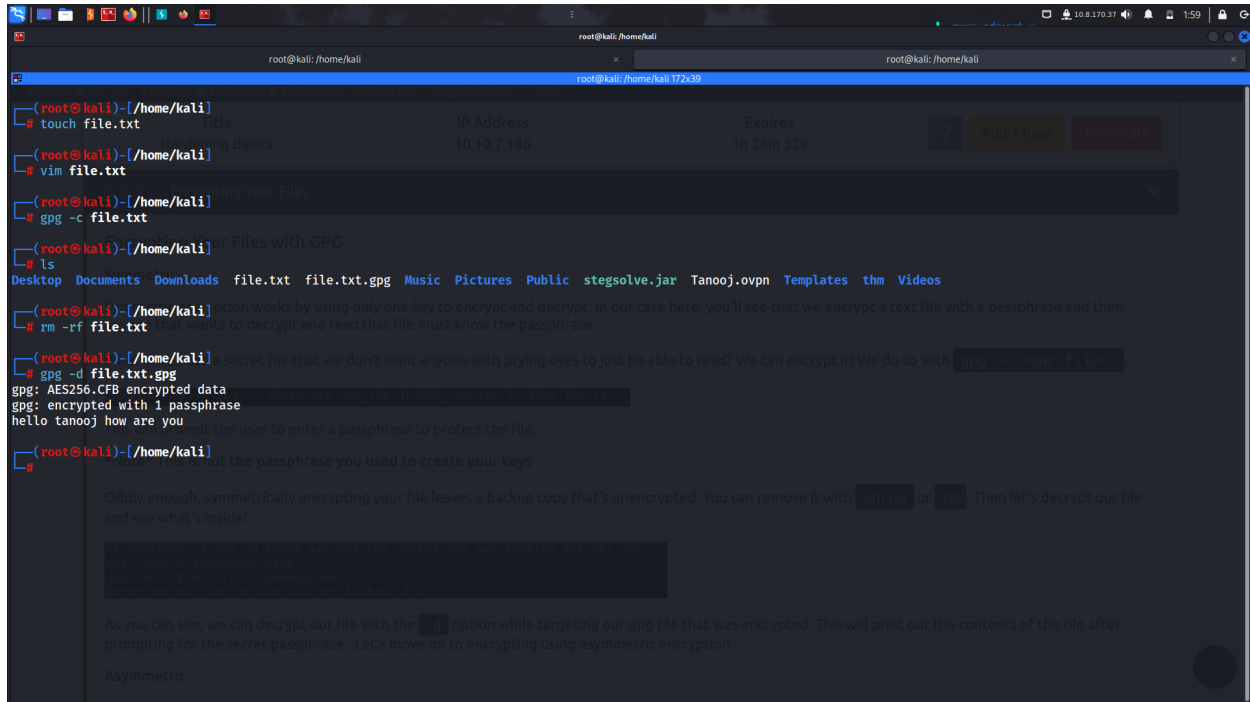
### **Symmetric**

Symmetric encryption works by using only one key to encrypt and decrypt. In our case here, you'll see that we encrypt a text file with a passphrase and then anyone that wants to decrypt and read that file must know the passphrase.

What if we have a secret file that we don't want anyone with prying eyes to just be able to read? We can encrypt it! We do so with **gpg -c <our\_file>** .

Oddly enough, symmetrically encrypting your file leaves a backup copy that's unencrypted. You can remove it with `shred` or `rm`. Then let's decrypt our file and see what's inside!

we can decrypt our file with the `-d` option while targeting our gpg file that was encrypted. This will print out the contents of the file after prompting for the secret passphrase. Let's move on to encrypting using asymmetric encryption.



```
(root@kali)~[/home/kali]
# touch file.txt
# vim file.txt
# gpg -c file.txt
# ls
Desktop Documents Downloads file.txt file.txt.gpg Music Pictures Public stegsolve.jar Tanooj.ovpn Templates thm Videos
# rm -rf file.txt
# gpg -d file.txt.gpg
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase
hello tanooj how are you
#
```

## Asymmetric

Asymmetric encryption works by using two keys - one to encrypt, and one to decrypt. The public key is used to encrypt the data while the private key is used to decrypt the data. So using the typical Bob and Alice example, let's say Bob wants to send Alice an encrypted file.

He would first encrypt the file using Alice's public key and then send the file away. Once Alice receives the file, she can decrypt it with her private key. The big takeaway here is that public keys can be shared, private keys should be kept private and held onto for dear life. NEVER SHARE YOUR PRIVATE KEY!

We'll need two users here. For this example, we'll use Nick and Spooky. Nick has a really super, secret file he wants to share but he doesn't want to have to share a passphrase. In order to do this, both parties need to have generated keys using the method from the previous task.

Since the public key is used to encrypt the data, both Nick and Spooky need to extract their public keys and send them to each other. We do that by navigating to the `.gnupg` folder and then `gpg --export -a -o <filename> .`

This will export the user's public key as ASCII armored output as the filename specified. In order to import the file, you need to be in that user's `.gnupg` directory or know the path.

As easy as that, we can import Spooky's key and he can import Nick's public key using the same command but changing the file name.

Now, let's say Nick wants to send Spooky an encrypted file. He has some really important document he needs to send. He will encrypt his document asymmetrically with `gpg -e <document> .`

## Creating an SSH Key Set

Are you always logging in to SSH with a password? Well, I'm here to tell you to stop. By far, the most secure way to login to SSH is with the use of secure keys. These can be generated by any user and the command will generate their own set of public and private keys to be used with SSH.

Like I said - any user can do this. Typically in the user's home directory, they can use the command `ssh-keygen` to generate their own pair of public and private keys.

I've gone ahead and done this as the user nick so you can see what the process looks like if you don't know already.

```

nick@harden:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/nick/.ssh/id_rsa):
Created directory '/home/nick/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/nick/.ssh/id_rsa.
Your public key has been saved in /home/nick/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:/eEjfSgTCRg/x3tIpeIl+J2nVLAap/Fo7Zlt4hkr5gY nick@harden
The key's randomart image is:
+---[RSA 2048]-----+
|      .      o      |
|      + o o o      |
|      . * X o .      |
|      . X / o      |
|      S % O .      |
|      .EO * o      |
|      +.O * o      |
|      +++ *      |
|      +..+      |
+----[SHA256]-----+
nick@harden:~$ █

```

## How to Disable Username & Password SSH Login

You'll only want to do this step after you've verified that your key exchange login works. Otherwise, you risk locking yourself or other users out of the system. To do this, you'll go to the `/etc/ssh/sshd_config` file and edit the following line

```
PasswordAuthentication yes█
```

to

```
PasswordAuthentication no█
```

which will completely remove password based logins. Again, BE SURE THAT YOU HAVE CONFIGURED THE KEY EXCHANGE TO WORK PROPERLY or this can completely lock you out from SSH.

## X11 Forwarding

You've connected to your workstation that has SSH enabled and you go about your work on the command-line. Everything is going great. But then you run into a problem. You need to run a program that only has a GUI. How would you accomplish that via SSH? That's what X11 Forwarding is for. X11 allows you to forward GUI application displays to your local environment (thought it has to have a GUI itself, right?). However, X11 has some flaws that make it dangerous to use. So let's look at turning it off.

### Turn off X11 Forwarding

To turn off X11 Forwarding is very simple. It's another setting in `sshd_config`. You'll want to find the line that says

```
X11 Forwarding yes
```

and change that to "no"

### SSH Tunneling

Let's say you're on your computer at work and your favorite streaming service is blocked. But you just wanna watch some in peace and not work, right? SSH Tunneling lets you do this. By forwarding the SSH connection to a computer or device running SSH that you own (most likely at home), you can browse this site at work. There's a few settings in the `sshd_config` that allow a user to accomplish this.

```
#AllowAgentForwarding yes
AllowTcpForwarding no
GatewayPorts no
X11Forwarding no
#X11DisplayOffset 10
#X11UseLocalhost yes
#PermitTTY yes
PrintMotd no
#PrintLastLog yes
#TCPKeepAlive yes
#UseLogin no
#PermitUserEnvironment no
#Compression delayed
#ClientAliveInterval 0
#ClientAliveCountMax 3
#UseDNS no
#PidFile /var/run/sshd.pid
#MaxStartups 10:30:100
PermitTunnel no
#ChrootDirectory none
#VersionAddendum none
```

The ones we're looking for are `AllowTcpForwarding`, `GatewayPorts`, `PermitTunnel`. All of these should be set to "no" in order to prevent SSH Tunneling and further harden the system and SSH.

---

## Configuring Improved SSH Logging

A log file is created any time someone logs in with a Protocol that uses SSH. So that would be SSH, SCP, or SFTP. By default, Ubuntu stores this log file in `/var/log/auth.log`. It looks something like this

```
Jun 24 18:48:39 harden sudo: pam_unix(sudo:session): session opened for user root by nick(uid=0)
Jun 24 18:49:03 harden sudo: pam_unix(sudo:session): session closed for user root
Jun 24 19:07:54 harden sudo:      nick : TTY=pts/0 ; PWD=/home/nick ; USER=root ; COMMAND=/usr/bin/vim -c :!/bin/bash
Jun 24 19:07:54 harden sudo: pam_unix(sudo:session): session opened for user root by nick(uid=0)
Jun 24 19:12:07 harden sudo: pam_unix(sudo:session): session closed for user root
Jun 24 19:17:01 harden CRON[6265]: pam_unix(cron:session): session opened for user root by (uid=0)
Jun 24 19:17:01 harden CRON[6265]: pam_unix(cron:session): session closed for user root
Jun 24 20:17:01 harden CRON[6371]: pam_unix(cron:session): session opened for user root by (uid=0)
Jun 24 20:17:01 harden CRON[6371]: pam_unix(cron:session): session closed for user root
Jun 24 21:17:01 harden CRON[6387]: pam_unix(cron:session): session opened for user root by (uid=0)
Jun 24 21:17:01 harden CRON[6387]: pam_unix(cron:session): session closed for user root
Jun 24 21:29:13 harden sshd[3193]: pam_unix(sshd:session): session closed for user nick
Jun 24 21:29:13 harden systemd-logind[953]: Removed session 13.
Jun 24 23:17:01 harden CRON[6402]: pam_unix(cron:session): session opened for user root by (uid=0)
Jun 24 23:17:01 harden CRON[6402]: pam_unix(cron:session): session closed for user root
Jun 25 00:00:01 harden CRON[6416]: pam_unix(cron:session): session opened for user root by (uid=0)
Jun 25 00:00:01 harden CRON[6416]: pam_unix(cron:session): session closed for user root
Jun 25 00:10:12 harden sshd[6423]: Accepted password for nick from 192.168.196.1 port 49246 ssh2
Jun 25 00:10:12 harden sshd[6423]: pam_unix(sshd:session): session opened for user nick by (uid=0)
Jun 25 00:10:12 harden systemd-logind[953]: New session 19 of user nick.
```

Neat. There's a few different levels of logging that you can find in the man pages of `sshd_config`.  
they are

- QUIET
- FATAL
- ERROR
- INFO
- VERBOSE
- DEBUG1
- DEBUG2
- DEBUG3

INFO is the default setting. This is one of the two we would normally care about. The other would be VERBOSE. To change the logging level is actually very simple and is an easy config change.

Navigate to `/etc/ssh/sshd_config` and look for the line that says `#LogLevel INFO`

We can uncomment that line and change it to any of the available levels above. And just like that, you'll now see more detailed logs in the `/var/log/auth.log` file.

## Mandatory Access Control

Mandatory Access Control (MAC) is a type of Access Control. It goes along with Discretionary Access Control, Role Based Access Control, and Rule Based Access Control. MAC is considered the strongest form of access control due to allowing more control over who has access over what. In a Linux system, there are multiple ways to implement MAC. Two of which being SELinux and AppArmor. We're going to take a look at AppArmor in this chapter and see how a system administrator or security enthusiast could harden their systems using MAC.

---

1. **Where are the AppArmor profiles located?**

ans: `/etc/apparmor.d`

2. **This directory includes partial profiles to be used in your own custom profiles**

ans: `abstractions`

3. **This punctuation mark is REQUIRED at the end of every rule in a profile**

ans: `,`

4. **This AppArmor mode enforces the profiles but also logs them**

ans: `audit`

5. **This command checks the status of AppArmor**

ans: `aa-status`

---