Assignment 8. SSH setup and use in applications

Laboratory

When you initially set up your host in the lab, you will be running an operating system, and will be able to connect to the outside world, but you won't be able to connect to the hosts of the other students in the class except in trivial ways. What you'd like to do is to be able to run processes on the other students' hosts. For example, you'd like to be able to log into a neighbor's host, and run a program there that displays on your host.

To do that, you need to set up an account on your neighbor's host, and vice versa so that your neighbor can log into your host and do the same. Unfortunately, the obvious ways to do that involve an initial step that exchanges passwords over the Internet in the clear. We'd like to avoid that.

In this laboratory, the class will divide into teams. Your team will assume that the other teams have all tapped the network connection and can observe the contents of all the packets going back and forth among all your team's computers. Your job is to set up your computers so that you can log into each other's hosts, without letting the other teams into your hosts.

Do not try to actually break into the other team's hosts; this is an exercise in defense, not offense!

Use OpenSSH to establish trusted connections among your teams' hosts. You want to make your logins convenient, so you should use ssh-agent on your host to manage authentication. That is, you should be able to log out of your host (dropping all your connections to the outside world), then log back in, type your passphrase once to `ssh-agent`, and then be able to use `ssh` to connect to any of your colleagues' hosts, without typing any passwords or passphrases.

You should also use port forwarding so that you can run a command on a remote host that displays on your host. For example, you should be able to log into a remote host, type the command `xterm`, and get a shell window on your host.

Keep a log of every step you personally took during the laboratory to configure your or your team members' hosts, and what the results of the step were. The idea behind recording your steps is that you should be able to reproduce your work later, if need be.

Homework

Use GNU Privacy Guard's shell commands to create a key pair. Export the public key, in ASCII format, into a file `hw8-pubkey.asc`. Use this key to create a detached signature for your submission so that the commands described below can successfully verify it.

Briefly answer the following questions.

1. Suppose the other teams really had been observing all the bytes going across the network. Is your resulting network still secure? If so, explain why, and explain whether your answer would change if you assumed the other teams had also tapped your keyboards and had observed all of your team's keystrokes. If not, explain any weaknesses of your team's setups, focusing on possible attacks by such outside observers.

2. Explain why the `gpg --verify` command in the following instructions doesn't really verify that you personally created the tar file in question. How would you go about fixing this problem?

Submit

Submit two files. The first should be a single gzipped tar file `hw8.tar.gz` containing the following files:

1. The file `hw8-pubkey.asc` as described above.
2. A copy of your lab log, as a file `log8.txt`.
3. The answers to the homework, as a file `hw8.txt`. This and `log8.txt` should both be ASCII text files, with with no more than 200 columns per line.

The second file `hw8.tar.gz.sig` should be a detached cleartext signature, in ASCII form, for `hw8.tar.gz`. It should use the key of `hw8-pubkey.asc`.

The following shell commands should work:

```
gunzip <hw8.tar.gz | tar xf -
mkdir -m go-rwx .gnupg
gpg --homedir .gnupg --import hw8-pubkey.asc
gpg --verify hw8.tar.gz.sig hw8.tar.gz
awk '200 < length' log8.txt hw8.txt
```

The `gpg --verify` command should say "Good signature". The last `awk` command should output nothing.

$Id: assign8.html,v 1.12 2010/11/19 21:40:14 eggert Exp $