Assignment 8. SSH setup and use in applications

Hardware prerequisite

For this assignment you will need a Seeed Studio BeagleBone Green Wireless Development Board (make sure you get the wireless version, not the plain one). You may wish to get the higher-priced Seeed Studio BeagleBone Green Wireless IOT Kit, as this is a superset of the basic unit needed for 35L, and is used by CS 111 this quarter (and likely in later quarters, though this is not guaranteed). These units are available from Seeed, Amazon, Digi-Key, Mouser Electronics, Verical, and other sources.

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 `xeyes`, and get a graphical window on your display. You will need to use `ssh -X` and/or `ssh -Y` when you use port forwarding.

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

On the SEASnet GNU/Linux servers, use GNU Privacy Guard's shell commands to create a key pair. Use GPG version 2. Export the public key, in ASCII format, into a file `hw-pubkey.asc`. Use this key to create a detached signature for your submission so that the commands described below can successfully verify it.

If you are creating a key pair on the SEASnet GNU/Linux servers, you may exhaust its entropy pool as described in Launchpad bug 706011. The symptom will be a diagnostic saying "It is a good idea to perform some other

action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy." Since you can't use the keyboard or mouse on the SEASnet servers, you'll have to use the disks, for example, by using the `find` command to copy every readable file to `/dev/null`; this is something that you can do in another session that is logged into the same machine. Please remember to interrupt the `find` once the key pair is generated, so that you don't tie up the server unnecessarily.

Briefly answer the following questions.

1. Suppose the other teams really had been observing all the bytes going across the network in your lab exercise. Is your resulting network still secure? If so, explain why, and explain whether your answer would change if (1) you assumed the other teams had also tapped your keyboards after you completed all client-server setup and had thereby obtained your team's keystrokes during later operation, or (2) you are booting off USB and you assume the other teams temporarily had physical control of the USB. If not, explain any weaknesses of your team's setups, focusing on possible attacks by such outside observers.
2. Explain why the `gpg2 --verify` command in the following instructions doesn't really verify that you personally created the file in question. How would you go about fixing this problem?

Submit

Submit five files, as follows:

1. The file `hw-pubkey.asc` as described above.
2. A copy of your lab log, as a file `log.txt`.
3. The answers to the homework, as a file `hw.txt`. This and `log.txt` should both be ASCII text files, with with no more than 200 columns per line.
4. A file `eeprom` that is a copy of the file `/sys/bus/i2c/devices/0-0050/eeprom` on your BeagleBone.
5. A file `eeprom.sig` that should be a detached cleartext signature, in ASCII form, for `eeprom`. It should use the key of `hw-pubkey.asc`.

The following shell commands should work:

```
mkdir -m go-rwx .gnupg
gpg2 --homedir .gnupg --import hw-pubkey.asc
gpg2 --homedir .gnupg --verify eeprom.sig eeprom
awk '200 < length' log.txt hw.txt
```

The `gpg2 --verify` command should say "Good signature". The last `awk` command should output nothing.

$Id: assign8.html,v 1.46 2019/09/24 05:36:59 eggert Exp $

```
Laboratory: SSH setup and use in applications

NOTE: Per the spec, this is a log of what I did in the lab so that I can reproduce the
results later and I've briefly noted down what I did and what happened. Trivial commands may
or may not be
explained.

# Partner: Pariya Samandi, UID: 205-092-357

#################### Steps taken to configure my BeagleBone

# First I completed the BeagleBone Setup Instructions listed here at home by myself:
https://piazza.com/class/k0zogkkf73r5dj?cid=463

# The following steps were done at YRL library while I was with my partner.

##### Connect to UCLA_WEB

sudo ssh root@192.168.7.2

connmanctl

Output of previous command:
Error getting VPN connections: The name net.connman.vpn was not provided by any .service
filesconnman

connmanctl> enable wifi

Output of previous command:
Error wifi: Already enabled

connmanctl> scan wifi

Output of previous command:
Scan completed for wifi

connmanctl> services

Output of previous command:
    BeagleBone-90F1      wifi_2cf7f106911b_426561676c65426f6e652d39304631_managed_psk
    SIL Network          wifi_2cf7f106911b_53494c204e6574776f726b_managed_psk
                         wifi_2cf7f106911b_hidden_managed_psk
    UCLA_WEB             wifi_2cf7f106911b_55434c415f574542_managed_none
    UCLA_WIFI            wifi_2cf7f106911b_55434c415f57494649_managed_none
    eduroam              wifi_2cf7f106911b_656475726f616d_managed_ieee8021x
    ubnt-ucla-yrl        wifi_2cf7f106911b_75626e742d75636c612d79726c_managed_psk

connmanctl> agent on

Output of previous command:
Agent registered

connmanctl> connect wifi_2cf7f106911b_55434c415f574542_managed_none

Output of previous command:
Connected wifi_2cf7f106911b_55434c415f574542_managed_none

connmanctl> quit

##### Make sure you have openssh-server and openssh-client installed

dpkg --get-selections | grep openssh

Output of previous command:
```

```
openssh-client                                  install
openssh-server                                  install
openssh-sftp-server                             install

##### Server Steps

# Generate public and private keys
ssh-keygen

Output of previous command:
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
2e:e8:1a:b7:fc:2b:1a:fe:07:a8:86:d4:50:12:14:44 root@beaglebone
The key's randomart image is:
+---[RSA 2048]----+
|+Eo              |
| . .             |
|  o              |
| .               |
|  o.    S        |
| ..... .         |
|o.o o.. .        |
|oo *....         |
|. ++=+o.         |
+-----------------+

# Create an account for the client on the server
sudo useradd -d /home/pariya -m pariya

Output of previous command:
Usage: useradd [options] LOGIN
       useradd -D
       useradd -D [options]

Options:
  -b, --base-dir BASE_DIR       base directory for the home directory of the
                                new account
  -c, --comment COMMENT         GECOS field of the new account
  -d, --home-dir HOME_DIR       home directory of the new account
  -D, --defaults                print or change default useradd configuration
  -e, --expiredate EXPIRE_DATE  expiration date of the new account
  -f, --inactive INACTIVE       password inactivity period of the new account
  -g, --gid GROUP               name or ID of the primary group of the new
                                account
  -G, --groups GROUPS           list of supplementary groups of the new
                                account
  -h, --help                    display this help message and exit
  -k, --skel SKEL_DIR           use this alternative skeleton directory
  -K, --key KEY=VALUE           override /etc/login.defs defaults
  -l, --no-log-init             do not add the user to the lastlog and
                                faillog databases
  -m, --create-home             create the user's home directory
  -M, --no-create-home          do not create the user's home directory
  -N, --no-user-group           do not create a group with the same name as
                                the user
  -o, --non-unique              allow to create users with duplicate
                                (non-unique) UID
  -p, --password PASSWORD       encrypted password of the new account
  -r, --system                  create a system account
  -R, --root CHROOT_DIR         directory to chroot into
```

```
  -s, --shell SHELL             login shell of the new account
  -u, --uid UID                 user ID of the new account
  -U, --user-group              create a group with the same name as the user
  -Z, --selinux-user SEUSER     use a specific SEUSER for the SELinux user mapping

sudo passwd pariya

# This gave me an error about the user not existing which made me realize the useradd command
wasn't executing properly (it displays the usage message -
https://piazza.com/class/k0zogkkf73r5dj?cid=539).

# We had to use a work around where we left off the option -d from useradd because useradd
will create the home directory in the same (default) location without that argument.

# Update: this may have been caused by copy/pasting the commands from the slides and getting
some undesired characters.

sudo useradd -m pariya

sudo passwd pariya              # Typed password "rip".

Output of previous command:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully

# The username for my account on my partner's BeagleBone is Stewart and the password is
"password".

# Create .ssh directory for new user
cd /home/pariya
sudo mkdir .ssh

# Change ownership and permission on .ssh directory
sudo chown -R pariya .ssh
sudo chmod 700 .ssh

##### How to Check IP Addresses
ifconfig

Output of previous command:
SoftAp0   Link encap:Ethernet  HWaddr c4:f3:12:7f:b9:f7
          inet addr:192.168.8.1  Bcast:192.168.8.255  Mask:255.255.255.0
          inet6 addr: fe80::c6f3:12ff:fe7f:b9f7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:89 errors:0 dropped:1 overruns:0 frame:0
          TX packets:132 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7516 (7.3 KiB)  TX bytes:20330 (19.8 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1738 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1738 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:150329 (146.8 KiB)  TX bytes:150329 (146.8 KiB)

usb0      Link encap:Ethernet  HWaddr c4:f3:12:7f:b9:f9
          inet addr:192.168.7.2  Bcast:192.168.7.255  Mask:255.255.255.0
          inet6 addr: fe80::c6f3:12ff:fe7f:b9f9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6851 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1840 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:1000
          RX bytes:2278098 (2.1 MiB)  TX bytes:338286 (330.3 KiB)

wlan0     Link encap:Ethernet  HWaddr 2c:f7:f1:06:91:1b
          inet addr:192.168.8.147  Bcast:192.168.8.255  Mask:255.255.255.0
          inet6 addr: fe80::2ef7:f1ff:fe06:911b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST DYNAMIC  MTU:1500  Metric:1
          RX packets:374 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1072 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:72120 (70.4 KiB)  TX bytes:231864 (226.4 KiB)

# The IP address of my BeagleBone is 192.168.8.147

# The IP address of my partner's BeagleBone is 172.20.10.6

##### Client Steps - Make logins convenient

# Copy your public key to the server for key-based authentication (~/.ssh/authorized_keys)
ssh-copy-id -i Stewart@172.20.10.6

Output of previous command:
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed

/usr/bin/ssh-copy-id: ERROR: ssh: connect to host 172.20.10.6 port 22: Connection timed out

# Connect to partner's BeagleBone's wifi network to avoid restrictions of public network
connmanctl
connmanctl> scan wifi
connmanctl> services

Output of previous command:
*Aa UCLA_WEB             wifi_2cf7f106911b_55434c415f574542_managed_none
    BeagleBone-90F1      wifi_2cf7f106911b_426561676c65426f6e652d39304631_managed_psk
    SIL Network          wifi_2cf7f106911b_53494c204e6574776f726b_managed_psk
    eduroam              wifi_2cf7f106911b_656475726f616d_managed_ieee8021x
    UCLA_WIFI            wifi_2cf7f106911b_55434c415f57494649_managed_none
                         wifi_2cf7f106911b_hidden_managed_psk

connmanctl> agent on
connmanctl> connect wifi_2cf7f106911b_426561676c65426f6e652d39304631_managed_psk

Output of previous command:
Agent RequestInput wifi_2cf7f106911b_426561676c65426f6e652d39304631_managed_psk
  Passphrase = [ Type=psk, Requirement=mandatory ]
Passphrase? # BeagleBone
Connected wifi_2cf7f106911b_426561676c65426f6e652d39304631_managed_psk

connmanctl> quit

ssh-copy-id -i Stewart@172.20.10.6

Output of previous command:
The authenticity of host '172.20.10.6 (172.20.10.6)' can't be established.
ECDSA key fingerprint is 63:88:ce:75:80:b2:22:58:f5:e3:36:f7:8b:97:a9:a5.
Are you sure you want to continue connecting (yes/no)? # yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys
Stewart@172.20.10.6's password: # password

Number of key(s) added: 1
```

```
Now try logging into the machine, with:   "ssh 'Stewart@172.20.10.6'"
and check to make sure that only the key(s) you wanted were added.

# Note that we were having an issue where I could ssh to my partner's BeagleBone but she
couldn't ssh/ping to mine, so after trying the workarounds for public networks listed on
Piazza like using
personal hotspots and the BeagleBone wifi networks for a long time, getting TA Shivam to try
with his own hotspot and he couldn't fix the issue, we came back to Boelter 3760 so sit on
chairs outside
the room and use the network CR3760-wifi. This resolved the issues we were having.

# Connect my BeagleBone to CR3760-wifi using connmanctl (details omitted for brevity).

# My new IP is 10.97.85.189, my partner's new IP is 10.97.85.190.

# Add private key to authentication agent (ssh-agent), so that I'm no longer prompted for the
passphrase for my private key when using ssh

eval $(ssh-agent)

Output of previous command:
Agent pid 4318

ssh-add

Output of previous command:
Identity added: /root/.ssh/id_rsa (rsa w/o comment)

# SSH to server and verify key was copied to ~/.ssh/authorized_keys
ssh Stewart@10.97.85.190

Output of previous command:
The authenticity of host '10.97.85.190 (10.97.85.190)' can't be established.
ECDSA key fingerprint is 63:88:ce:75:80:b2:22:58:f5:e3:36:f7:8b:97:a9:a5.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.97.85.190' (ECDSA) to the list of known hosts.

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Oct  7 18:51:22 2018 from 192.168.6.1

cat ~/.ssh/authorized_keys

# Test run a program on partner's host that displays on my host
vi hello.txt         # type "Hello World"

# Test port forwarding enables running a command on a remote host that shows a graphical
display on my host
exit
exit
sudo ssh -X root@192.168.7.2
ssh -X Stewart@10.97.85.190

Output of previous command:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

```
Last login: Sun Oct  7 16:49:01 2018 from 10.97.85.189

firefox

# We see the Firefox browser open up in XQuartz on my macbook laptop and a lot of debug
messages from Firefox in the terminal, demonstrating that the configuration I did for X11
forwarding based on
the Piazza instructions was done correctly. Note that using option -X for ssh was needed in
both ssh sessions (laptop to BeagleBone and BeagleBone to partner's BeagleBone).

#################### Steps taken to configure my partner's BeagleBone (if any)

N/A
```

```
Homework: SSH setup and use in applications

NOTE: Per the spec, this is a log of what I did in the lab so that I can reproduce the
results later and I've briefly noted down what I did and what happened. Trivial commands may
or may not be
explained.

# The GNU/Linux server used was lnxsrv06.

# Generate a key pair with the GNU Privacy Guard's commands (choose default options when
prompted)
gpg2 --gen-key

Output of previous command:
gpg (GnuPG) 2.0.22; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection?
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Stewart Dulaney
Email address: sdulaney@ucla.edu
Comment: Turkey
You selected this USER-ID:
    "Stewart Dulaney (Turkey) <sdulaney@ucla.edu>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.   # 6BdwGjrdUh

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key FE75A706 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
pub   2048R/FE75A706 2019-11-26
      Key fingerprint = 0CEC 608C 62B2 9C48 542E  EA30 2A96 275D FE75 A706
```

```
uid                  Stewart Dulaney (Turkey) <sdulaney@ucla.edu>
sub   2048R/78F2BDAB 2019-11-26

# Note the step to increase entropy was not needed in this case.

# Export public key, in ASCII format, into hw-pubkey.asc
gpg2 --armor --export sdulaney@ucla.edu > hw-pubkey.asc

# Copy /sys/bus/i2c/devices/0-0050/eeprom from BeagleBone to my laptop
scp root@192.168.7.2:/sys/bus/i2c/devices/0-0050/eeprom ~/Downloads

Output of previous command:
Debian GNU/Linux 8

BeagleBoard.org Debian Image 2016-10-20

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

eeprom                                                        100%   32KB  34.3KB/s
00:00

# Copy eeprom to lnxsrv06
scp ~/Downloads/eeprom stewart@lnxsrv06.seas.ucla.edu:/u/cs/ugrad/stewart/Assignment8/hw/

Output of previous command:
eeprom                                                        100%   32KB   1.8MB/s
00:00

# Use the private key you created to make a detached clear signature eeprom.sig for eeprom
gpg2 --armor --output eeprom.sig --detach-sig eeprom

Output of previous command:

You need a passphrase to unlock the secret key for
user: "Stewart Dulaney (Turkey) <sdulaney@ucla.edu>"
2048-bit RSA key, ID FE75A706, created 2019-11-26       # 6BdwGjrdUh

# The following shell commands should work and the gpg2 --verify command should say "Good
signature"
mkdir -m go-rwx .gnupg
gpg2 --homedir .gnupg --import hw-pubkey.asc

Output of previous command:
gpg: keyring `.gnupg/secring.gpg' created
gpg: keyring `.gnupg/pubring.gpg' created
gpg: .gnupg/trustdb.gpg: trustdb created
gpg: key FE75A706: public key "Stewart Dulaney (Turkey) <sdulaney@ucla.edu>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)

gpg2 --homedir .gnupg --verify eeprom.sig eeprom

Output of previous command:
gpg: Signature made Tue Nov 26 14:27:53 2019 PST using RSA key ID FE75A706
gpg: Good signature from "Stewart Dulaney (Turkey) <sdulaney@ucla.edu>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 0CEC 608C 62B2 9C48 542E  EA30 2A96 275D FE75 A706

# Note the gpg2 --verify command says "Good signature" as desired.

########## Question 1
```

Suppose the other teams really had been observing all the bytes going across the network in your lab exercise. Is your resulting network still secure? If so, explain why.

Yes, the network is still secure because we are using SSH (secure shell) to send data over the network (for remote login and shell access between the BeagleBone's).
- This means that even if an attacker observes the bytes going across the network, the messages sent between our BeagleBone's will be encrypted with a symmetric encryption key (session key).
- I was curious if this session key could be sniffed over the network but I learned that the client and server use the Diffie-Hellman algorithm for key exchange, which allows them to jointly establish
a shared secret key over an insecure channel.
- Therefore, without the session key, an attacker would not be able to decrypt the communications between our BeagleBone's.
- It should be noted that (symmetric) secret key used to encrypt communications is completely separate from the (asymmetric) public key/private key pair used for client authentication, and the
symmetrical encryption is established before client authentication.
- Because ssh-copy-id uses SSH under the hood (with password authentication in our case), the attacker cannot decrypt the messages containing the password used for authentication and public key being
copied to the server due to the same symmetric encryption reasoning.

Explain whether your answer would change if:

(1) you assumed the other teams had also tapped your keyboards after you completed all client-server setup and had thereby obtained your team's keystrokes during later operation

Yes, the network is still secure because the only sensitive info that would be compromised is the passphrase for the private key used for client authentication, which doesn't give the attacker access
to anything without the private key file itself.
- The password for my account on my partner's BeagleBone would not be compromised because after completing client-server setup (namely, the step copying my public key to the server), key-based
authentication is used instead of password authentication.
- All communications over the network would still be encrypted per the explanation above.
- Although confidentiality is compromised because the observer knows commands we typed after setup and the passphrase for our private key, the network itself is still secure because an attacker can't
decrypt communications or login to our hosts.

or (2) you are booting off USB and you assume the other teams temporarily had physical control of the USB

If not, explain any weaknesses of your team's setups, focusing on possible attacks by such outside observers.

No, the network is not secure in this case because if attackers had control of a USB I am booting off they could copy my public and private keys from ~/.ssh and therefore be in control of my private
key.
- A possible attack could be that with a copy of my private key, they could impersonate me and SSH into my account on my partner's BeagleBone using key-based authentication.
- Because the passwords for user accounts on my host are hashed before being stored (irreversible except by brute force), the attacker could only SSH into my host with password authentication if they
brute force tried hashing different passwords with the same algorithm and got lucky and produced the same hash (thereby learning the plaintext password). However, this is unlikely.
- The attacker could also copy my partner's public key from ~/.ssh/authorized_keys in their home directory on my host, but this would not give them access to that account on my host (they'd need the
private key), nor would it give the attacker access to my partner's host.

########## Question 2

Explain why the gpg2 --verify command in the following instructions doesn't really verify that you personally created the file in question.

The command we ran was:
gpg2 --homedir .gnupg --verify eeprom.sig eeprom

The output we saw was:
gpg: Signature made Tue Nov 26 14:27:53 2019 PST using RSA key ID FE75A706
gpg: Good signature from "Stewart Dulaney (Turkey) <sdulaney@ucla.edu>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the owner.
Primary key fingerprint: 0CEC 608C 62B2 9C48 542E EA30 2A96 275D FE75 A706

The gpg2 --verify command only checks that the specified signature (.sig file) was created using the private key corresponding to a given public key (in this case the one we imported to our public
keyring, hw-pubkey.asc) and that the specified document has not been modified. The "Good signature" message indicates these checks have passed.

However, the command doesn't really verify that I personally created the signature file in question. This is because anyone, including a malicious party performing a man in the middle attack, can
generate a public/private key pair using gpg2 --gen-key with my name and information as the user ID. For example, if said malicious party intercepted a message from me to my partner containing a
document/signature/my public key and replaced it with a modified document/signature generated with their private key/their public key (with my name in the user ID), when my partner ran the
gpg2 --verify command it would still say "Good signature" and that it was signed by me (when in fact it was not).

How would you go about fixing this problem?

In order to address this problem, one way would be to publish my public key on a public website like the GNU Project does with their keyring as a way to verify the signature of Coreutils. However,
this doesn't really verify identity of the author either as you have to assume they are in control of the website and that it has not been tampered with. A better solution is a trusted third party,
such as a Certificate authority (CA). This party validates a person's identity and either generates a public/private key pair on their behalf or associates an existing public key provided by that
person to that person. Once they verify someone's identity, they issue a digital certificate that is signed by the CA, which can be used to verify a person associated with a public key when requested.
As long as the third party is trusted, this solves the problem because a receiver of a message from me can verify a public key is actually mine, and then check that a document was signed by me using
that public key. Another way people address this issue is by marking keys as trusted themselves (say I know my partner's key so I decide to trust it), or by trusting keys within your web of trust
(Source: GPG Manual).

```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2.0.22 (GNU/Linux)

iQEcBAABAgAGBQJd3abpAAoJECqWJ13+dacGOYgH+wRtmGlkX/rh32XhTff7DbLL
DNDuv9lS+luiADrI46oZnyLU01Ksf1UEE4+qkjWphtRDiM6rbm1EoMOzOZC/FTKE
kfISEFx5KHsgh8YJgIocrSPDIHRcCrDoL60Qzze5qedwpnJL/O6RpBzPuTjHH9Tk
09yaQIzv2/W7DHyry6jdPQAk7P1PrX8ZmtK6+G5QQfEAW3oc3S5K0+R7y0tZaDrx
ojQk5eAhLrL3I7uGunaPJEbzLBlZCuGj84xVS3nb4+PBurB9il/Z6qUo/6cdesdC
qFEYiF9zi1IjwIrf3toOCGzY9sYtiU27njHlqlZNbnJRHIQ3WZGghPxyUlw9j5E=
=fgJu
-----END PGP SIGNATURE-----
```

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQENBF3doMUBCADHqsgNpkEQUcmS/AcfUUHE9hgNdhg+/7UrnzkZsjDl+Ma46Gfa
IwJQjXjv5rdZ3QDGSxl8dUSl7j2FVSKyRIrakxy32BkZ+ULScPOIvgvcX2T3kqbU
94AG4hB8uQeoTtoknzV4Nv3lA/jitxLqiD6PmnRwyS7+dJtahr1g0YPWfrOdii46
B1duSfhQ2+utQ//YdfkFvKumFaIvY256TM6SMYDcDRVdZjNV8d0Skic7W9FIb5eN
uhVNOrjqcml2bCDN2n/MAcLUq21sDYrAUPqHcpYvV05ib4Sg8sNbFLQSXYww3WPn
wbnxKbYbg0EOzwHTAGBDFDM/CoharMCRmZKDABEBAAG0LFN0ZXdhcnQgRHVsYW5l
eSAoVHVya2V5KSA8c2R1bGFuZXlAdWNsYS5lZHU+iQE5BBMBAgAjBQJd3aDFAhsD
BwsJCAcDAgEGFQgCCQoLBBYCAwECHgECF4AACgkQKpYnXf51pwb7ewgAgM2lto0i
KA2c2V79vo14dnRe4MO4KePRRsYgi5LZdO8w7EPrKwjv5Y1HU7AQFCuolE1LnfLL
dG6i0xPUNzSqnYuUOoweh6gFlFzOEn5qLzFM/NfMvfjyFZYH6iX7H/HxOUtGRH8O
XeQ0sPNpesrC056hFdTppdI4NAmJgDn41nniNgcu0G2C8K/01xWl/77Ge4A4UCGW
o2hK5TpQW7BqQ8/HZRh0ayXgs1gcR7zaYmQe0L5Aza+PnMSZ0fuoZ+8dvWOb47xg
GNoazi8nIe1sLj80UJdvbyb5e6Dv3bNB1K0tjvJI0Bhz0+F0k+fH4cYlfkODNIdf
aR/6c7trSWEwVLkBDQRd3aDFAQgA13u8DOjo9inJ3hi1v4SZ77SgxzbxLXkQxQD6
FMljc8YIjGzAXX023n43cfT0kkjfHjutY0IoMZl5lhMj0iaKYcSsSRwvOUUnaaf1
8RaH9ARODcX7r7J80H8nPIM/i6Q0HQXYwv5lbkL2Gb8w+BdGju9If9iWVkOijO3D
DKGbifsuupxi1UnZKhv35po6jFsIk6OfW5u1tG/r5EwSb/a4qKZCxgmLNVjmA+mw
c22XvDkF4aPVO8ID19kgFODPNrhEWY4Hdif5MGS5tihYT+nSkEdhzzrUV+oQu/L5
bVmYSyepVi11I2CfW4CizIHl2ER6kVFauo+0tzOe7sYG+HZ40QARAQABiQEfBBgB
AgAJBQJd3aDFAhsMAAoJECqWJ13+dacGpTIIAMOJNPuGLp03teBAbFG3SmaquJxM
RBg9Sd8I0tHAq5FkXE/Zw1h6uCHZhaT+Shk11pzQ05YSsoecMN4lTv/7Aw1uZfzL
HzKaC5Hk+UwdtOYjowlE9tNCTqD11l/JP63bIrgcaVr+N0aMKHpHLEhZOBOTG7/+
+6UeFnikmR6fgE3EGZmsWQ8Rko+hTEE1EPHA/E8oNXopGRpdWO2IBalmnLgCrkK1
xD+NiUF/i1ZKq5GLX5NozRZCqqYUm4t7b7HATsS+uDEXumNSBK5srsOdQRy/ZaZU
nxTPq7QdulX3cnqoOHZtxKsV9f/CnqIPbyUsg/mSb+5o8/EAxYojLxlLVRs=
=6f3V
-----END PGP PUBLIC KEY BLOCK-----
```