This document lives here:
http://inst.eecs.berkeley.edu/~ee122/fa12/project3/guide-to-plug.pdf

Guide to your Plug Computer

UC Berkeley, EE 122, Fall 2011

Version 1

This document is a step-by-step guide to using the plug computer for Project 3. It is divided into the following sections.

You should read the first few sections carefully before using your plug. Using your plug without reading these instructions might be inordinately hard. There is a set of advanced tools listed at the end. Reading about these might be extremely beneficial.

1. Prerequisites

Check to make sure you have been provided with the following:

1. A plug computer with
    a. Power cable (black cable)
    b. Ethernet cable (white cable)
    c. Serial port connector and wall socket connector (ignore these)
2. An 4GB USB stick

For doing this project, you need to have the following

1. Access to AirBears wireless network.
2. A place to hook up for power.
3. [Optional but recommended] Infinite passion for debugging real networks.

2. Preparing the USB stick

The plug computers have been setup so they boot from the supplied USB sticks. The USB sticks you are provided with do not however have an operating system on them. You therefore need to follow the steps below to set your USB stick up. It is possible, especially in cases where you experience a sudden loss of power, that the USB stick gets corrupted. In this case you would need to follow these instructions again. Note, that following these instructions means that your USB stick will be wiped clean, and you will be starting from the beginning.

a. Download the image

1. Download the compressed image file - http://inst.eecs.berkeley.edu/~ee122/fa12/project3/new-img-4g-v2.img.bz2
2. Check whether its MD5 hash is **64a19634846c4392bde826c83ca7c3f7** .
   (Instructions to check MD5 hash - https://help.ubuntu.com/community/HowToMD5SUM)
3. Uncompressing it using `bunzip2 <path to image>`, will give you the image file **plug-image-v1.img** to write to the USB stick.

Step 2 is optional but recommended.

b. Burn the image to USB stick

Windows

1. Install **Disk Imager** from https://launchpad.net/win32-image-writer/+download
2. Insert USB stick, and note the drive letter assigned to it.
3. Start Disk Imager, select the target device and the image file, and click “Write”.
4. Take out the USB once the writing is done.

Ubuntu

1. Install **usb-imagewrite** package, by `sudo apt-get install usb-imagewriter`
2. Insert the USB stick.
3. Open Applications -> Accessories -> Image Writer. KDE users will find this in Applications -> Utilities -> Image Writer.
4. Select the image file, then select the correct device, and click “Write to Device”.

5. Take out the USB once the writing is done.

Mac OSX

1. Open Applications -> Utilities -> Terminal. Run ”`diskutil list`” in the Terminal to get the current list of storage devices attached to your system. They will be of the form **/dev/diskN**, where N is an integer.
2. Insert the USB stick. Run ”`diskutil list`” again and see what new storage device got added. This is the device file that represents the USB stick. On most systems that do not have any other USB drives attached, this will be **/dev/disk1**.
3. Download this script - http://inst.eecs.berkeley.edu/~ee122/fa11/project3/write-to-usb
4. Make it executable using “`chmod +x write-to-usb`”, and run it as

   ```
   sudo ./write-to-usb <path to device> <path to image.img>
   ```

   where <device> is the device found in step 4. For example, if image file is in the current working directory, then

   ```
   sudo ./write-to-usb /dev/disk1 plug_image_v1.img
   ```
4. This process should take less 10 minutes. You will see the progress printed on the terminal in terms of number of bytes written (out of 2GB).
5. Take out the USB stick when the script says that it is safe to do so.
6. At any point, if you get a prompt saying *“The disk you inserted was not readable by this computer [Initialize / Ignore / Eject]”*, always choose “*Ignore*”. **Never select “*Initialize*”, or your the data on the USB stick may get corrupted.**

3. Working with the plug

In this section, we are going to explain how to turn on the plug and connect to it, such that you can use the plug as a router to browse the Internet. For this to be possible, you need to have access to the AirBears wireless network.

[Optional] It is relatively easy to have the Plug computer associate with another wireless network - such as the one you may have at home. It is also possible to turn the Plug into a wireless access point and connect to it wirelessly and get access to Internet via the wired Ethernet port - however, these configurations may create unexpected challenges that may be difficult for the TAs to support. We therefore discourage these and while we will expose some hints about getting this done - we would not support these modes.

a. Powering up a Plug

1. Plug the USB stick to the plug.
2. Now, connect the plug to power (plug in the plug!). Within about 10 seconds, the light will turn blue and then turn back off.
3. Once the blue light has turned off, push the power switch away from the lights, thus turning the plug on.
4. The blue light will start flashing, and then within a couple of minutes the flashing blue light should turn into a steady red light. This means that Linux kernel from the USB stick is booting, and your system works! If you instead see a flashing red light, the Plug cannot recognize the operating system on the USB stick. Unplug the plug (!) and the

USB stick and try again from step 1.

5. Within about 5 minutes, the steady red should turn into a steady blue. This means Linux has booted and your plug is ready to be used. You can now connect to the Plug computer.

   To summarize the sequence,

   connect power, wait for long blue blink, turn it on, flashing blue, steady red, steady blue

   If you repeatedly fail to get this sequence and always end up in a **flashing red**, then you may have not have been able to write the image correctly. Write the image again, and then try it out again. If the plug still doesn't turn on correctly, contact **Panda** straightaway (or consult other students that got it working, especially ask them to wire the image on the USB drive for you).

b. Using the plug

1. After the plug has started (that is, steady blue), it is configured to automatically connect to AirBears using its wireless interface.
2. Connect its Ethernet / LAN port to that of your laptop, using the provided Ethernet cable. The plug runs a DHCP server on the Ethernet port, so your laptop will be given an IP address of **10.1.1.*** . The plug is now acting as a router running NAT - your laptop is a client behind the NAT.
3. **Turn off your laptop's wireless network.** This forces your laptop to try to reach the Internet through the plug.
4. Open up your browser and try connecting to a website (say *www.google.com*). You should be taken to the usual AirBears authentication page. After logging in you should be able to do all your browsing through the plug computer.
5. If you have just booted up your plug, give it about 4 - 5 mins to connect to AirBears.
6. **AirBears in Soda Hall can be rather flaky at times. If you find that your connection is not working, try running reset-connection, and see if it works. Also some of the more advanced tools at the end might help with diagnosis.**

c. Turning off the plug

Push the power switch to the left (away from the lights) repeatedly (and fast) until the **steady blue** light turns to a **steady purple/violet**. Within 10 seconds, it will turn **steady red**, which means that the plug has started to shut down. Wait for the light to turn off. Then unplug the Plug from power. You need to unplug it even if you want to turn on the plug again, otherwise it will not turn on.

Unplugging the plug without shutting it down first can lead to data corruption. Hence, **it is strongly advised that you shutdown the plug before unplugging it.**

4. Software environment

In this section we will explain the software environment of the plug. The plug is primarily meant to be used as a terminal-based development environment, much of what we discuss relies on using the terminal (the sooner you get used to it, the better!).

a. Logging into the plug

The internal IP address of the plug is **10.1.1.1** . It has two users: "**root**" and "**user**". The password for both is "**ee122**". You should be able to login into the plug from a terminal as

"**ssh root@10.1.1.1**"

If your system supports it, you can use the **-X** option (**ssh -X root@ …**) for enabling *X Forwarding* (see in the *Tips and Tricks* section for more information).

You may also use "terminal multiplexing" tools like "tmux" or "screen" - (see in the *Tips and Tricks* section for more information).

Once you have successfully logged into the plug, **you are strongly advised to change the password of both users**. Use the command `passwd <username>` to do it. Unless you do that, anybody can take your plug and log into it!

Besides SSH server, the plug runs a VNC server on port 5901 as session 1, which you can connect to using a VNC client[1]. Additionally, the plugs runs a web-based VNC viewer on port 5801, which you can connect to as http://10.1.1.1:5801.

b. Software environment

Project Files

All the files related to the Project 3 are located at **/root/pox/ext**. You will find the following files in this directory.

1. ***firewall.py***: This is the file that you are going to modify to add firewall rules. This file contains the Firewall class with the handler functions for the event *ConnectionIn*, *DeferredConnectionIn*, and *MonitorData*. Refer to the Project 3 specification for more details.
2. ***firewall.py.original***: This is a copy of the default *firewall.py* provided with the plug. This may come in handy while debugging your own *firewall.py*.
3. ***banned-ports.txt, banned-domains.txt, monitored-string.txt***: These are sample files for doing Project 3 Part 1

Useful Commands

[1] https://help.ubuntu.com/community/VNC/Clients

1. ***cycle-pox***: This command restarts POX. After modifying *firewall.py*, you need to run this command to make POX use the new *firewall.py*.
2. ***fwlog***: This command shows the running log of debug statements logged in the file *firewall.py* (using `log.debug(...)`).
3. ***poxlog***: This command shows the running log of all the debug statements POX (including that of *firewall.py*). Besides the debug statements from *firewall.py*, you will find statements of the form "*Pushed **X** rules for <Connection … >*". This signifies how many rules have been pushed by the POX to the OpenFlow switch. If X > 0, then the corresponding connection was allowed, otherwise it was denied.
   While poxlog serves an important purpose, errors aren't normally logged to poxlog. For this iteration of the project we are logging errors at /tmp/pox-run and on tmux (explained later). Make sure /tmp/pox-run has no errors before handing in your code.
4. ***reset-connection:*** Resets the connection to AirBears (if you use your own wireless network you'd need to edit this).

A typical way to use the scripts is as follows. Have multiple ssh sessions to the plug (10.1.1.1) open, each in a different terminal window. One or two windows would run *poxlog* and/or *fwlog*. Another window is to be used to editing firewall.py. Every time you edit firewall.py, you run *cycle-pox* to make POX use the modified firewall.py. Test whether the firewall works by generating appropriate connections. For example, if you have written a rule in *firewall.py*, that should block connection to *www.amazon.com*, then try browsing *www.amazon.com*. Or, if you have tried to block connections to SSH port, try connecting to a publicly visible SSH server like *ftp.mozilla.org*. If the behavior of the firewall is not as expected, then check the output of POX and/or firewall, in the window(s) running *poxlog* or *fwlog*. Accordingly edit the *firewall.py*. And so on.

While poxlog provides a valuable insight into logging message specific to your application, it does not trap exceptions. We have

The directory /root/pox/pox contains the source code of POX. You can safely ignore all the files in this directory, except the directory /root/pox/pox/lib/packet. You will find the details of the data structures representing an Ethernet, IP and TCP packets in this directory. As before, refer to the Project 3 specification for more details.

The directory /root/pox/ is under Git version control. Whenever we push out updates, you can update your environment by running "`git pull`" . Every time you make a image, make sure you do an update.

5. Tips and Tricks

- **Make sure you take regular backup of the files you modified in the plug.** You do not want to lose your work if the data in the USB stick gets corrupted and the stick needs to be re-imaged. Use of some form of source versioning system is highly recommended.

- If you are testing whether a website is blocked through through the firewall or not, close your browser and reopen it. Browsers maintain persistent connections and do a lot of caching of website which may lead to very non-deterministic behavior. Restarting the browser (or opening up a new private mode window) forces the browser to fetch the intended page over a new connection, and hence behaves much more predictably.

- You can use the standard Debian style "*sudo apt-get install …*" method for installing anything on the Plug. Its your own environment, go ahead and customize it!

- If you want to use GUI-based applications (GVim, XEmacs, Wireshark, etc), you can do one of the following methods.
    - As mentioned before, you can connect to the VNC sessions already running in the plug. You should be able to run GUI-based applications in a VNC session. VNC sessions persist even if you get disconnected - when you reconnect, all your windows will stay open as it was when you disconnected. You can even share the same VNC session with your partner.
    - Alternatively, you can setup X11 Forwarding. If you are on any Unix-based system (Ubuntu, Mac OSX, etc), all you need to do is SSH using the -X option. Then you can simply start any GUI-based application, and it will appear as a native window in your local system. If you are on Windows, install *Xming* and *PuTTY*[2], and you are good to go!

  One caution though. These plug computers do not have much resources (CPU, RAM, etc) . Hence, if you running something that is heavy on the resources (like full gnome session, eclipse, etc), then not only will the the application run slowly, the performance of the firewall may be affected. You DO NOT want that to happen. Hence, use this carefully. Running just your favourite text editor (e.g., nano, gedit, gvim, emacs) should be perfectly fine.

- Instead of editing the code on the plug, you can edit the code in the local system and copy the necessary files over to the plug for testing. Use rsync (command line), scp (command line) or FileZilla (GUI) for copying between the Plug and other systems.

- Terminal multiplexers like `screen` or `tmux` are pre-installed on the Plugs. If you want to run something in a terminal (for example, after logging into the plug) that you would like to continue running even after you get disconnected, then these tools are helpful. If you start either of these in your terminal window, and then run commands in them, they will continue to run even if you get accidentally disconnected and you can later re-connect to it. Also - use of these tools allows one to switch back and forth between virtual terminal

[2] http://blog.nth-design.com/2010/05/19/x11-putty-xming/

sessions (like different windows on a GUI) and that's very handy. See the manual pages for these commands for details.

- If you want to take a direct look at all packets going in and out of the plug, you can use *Wireshark* and/or *tcpdump*. They are already installed. There are number of tutorials around for using them.

Some Advanced Tools

- **man**: man is a unix utility for displaying manual pages. Generally typing in something like "*man <command>*" will provide you with information about the command. Using *man* pages is an essential skill for surviving this project, since the TAs and even *gasp* Scott know less about the semantics of some of these things than the man pages.
- **ifconfig**: You can use ifconfig to look at whether the DHCP server on AirBears has actually given you an IP address. ifconfig will show all available interfaces, so it is useful to know that the wireless interface is called mlan0. You can find more information about ifconfig in the man pages.
- **iwconfig**: Similar to ifconfig, iwconfig provides information about wireless network interfaces including what network you are associated with (the SSID). iwconfig also provides a way to associate with another network, and might be very useful if working from home.
- **dhclient**: dhclient is the DHCP client for the particular flavor of Linux we are using. Running dhclient mlan0 will cause the plug to renew its DHCP lease. Read the man pages to find other options that might be useful.
- **tmux**: Pretty much everything interesting on the plug is being run on tmux a "terminal multiplexer". If you run *tmux attach*, you can see output logs from *pox*, the program that is actually executing your firewall code, *dnsmasq*, the program forwarding DNS requests and providing DHCP services for the plug. You can find more information about tmux online, http://blog.hawkhost.com/2010/06/28/tmux-the-terminal-multiplexer/ is a good source of information. Some things to keep in mind:
  - As currently provided, tmux has ctrl-B set as its prefix.
  - If you open tmux and get stuck, the easiest way to leave (while leaving everything intact) is typing Ctrl-B + D, i.e. first press the ctrl and the B key together, then release them, and type D.
- **dig**: *dig* is a tool that can be used to carry out a DNS lookup. Often the problem one has with AirBears involves not contacting the correct DNS server (or any DNS server really). Using dig might be a good way to check if this is the case. Look at the man pages for information how.
- **curl**: As you try and experiment with the firewall, you will notice that browsers often cache content, and hence don't always get content over the network. An easy way to prevent this is to use the tool curl, which in addition to not caching your content allows you to examine HTTP headers and other similar things.
- **rsync**: *rsync* provides an easy way for you to keep files in "sync" between two directories. rsync can connect over ssh, and is useful for either taking backups, or doing your work on a computer and transferring data around. Again look at the man pages for information.

- In general a big part of this project is learning how to survive in unknown environments. You should use Google to find information, you can install anything you want, especially if it helps you with the project. You really should take the time to explore some of these tools, and figure out what is useful.