CS 470: Unix/Linux Sysadmin
Spring 2025 Lab 2
FreeBSD and e-mail with `sendmail` and `dovecot`

Things from prior labs that are required to begin this lab:

- lab 1b: working certificate authority and LDAP server

Things in this lab that are on the critical path for upcoming labs:

- getting FreeBSD up and online
- getting your mail server accepting mail and delivering it to your inbox
- doing the errata for this lab

This lab gets our hands dirty with the far-and-away most prominent free-and-open-source BSD-derived operating system, FreeBSD. Whereas OpenBSD has always focused on being the most secure operating system (see http://www.openbsd.org/goals.html), FreeBSD has always been focused, in their own words, "to provide a stable and fast general purpose operating system that may be used for any purpose without strings attached." The language I quoted used to be found on this page, and has been removed since I first reference it in this lab years ago, but references to it and its general intent remain.

OpenBSD has always quite unabashedly only catered to power users and hackers, and famously expects it users to know what to do with it … which means the operating systems mostly get easier from here. Pat yourself on the back, you're over the proverbial hump.

To further its design goals, FreeBSD's codebase is much larger than that of OpenBSD, and the amount of supported hardware and third-party software is much greater. There are also a lot more creature comforts all around. FreeBSD makes less design assumptions about a user's level of skill or threshold for pain, and has a much larger community of developers and users – notably, FreeBSD code has been extensively used in the development of macOS, the Sony PlayStation line of gaming consoles, under the sinister hood of Google's ~~search~~ surveillance engine, and inside Juniper routers, and NetApp network storage devices ... the latter two obviously because of FreeBSD well-established reputation for having the fastest TCP/IP stack, of any operating system. For many years, the fastest and highest-traffic internet sites ran FreeBSD, until the time came where producing statistics was no longer feasible.

Once upon a time, another surveillance and mind control company had a program to bring the Linux kernel's famously bad IP stack up to parity with FreeBSD's.

https://www.theregister.com/2014/08/07/facebook_wants_linux_networking_as_good_as_freebsd/

That says it; for over thirty years now, if you want to push traffic as fast as you can, or you want to take a derivative source base private, you use Berkeley Unix (BSD), and that's that.

Back to the greater scheme of things. We have plumbed IP and name service for our little private network in lab 1, and provided encryption and shared authentication infrastructure in lab 1b. Now we'll set up e-mail services, to see what setting up the first killer application of the internet looks like, and to provide centralized collection of periodic diagnostic jobs as we delve further into system internals. FreeBSD is going to be our mail server, to help us aggregate all this data.

‼ IMPORTANT NOTE: after setting up this lab, you will want to start leaving your VMs on overnight, at least some of the time. A lot of `cron` jobs run during the late hours, by design, and we want you to see their output! If you'd rather not leave your computer on overnight, some students in the past have made nighty `cron` jobs run by day when they knew they'd be up and with their computers on. By all means read `man cron` and tinker, but you're on your own here if you do this … just make sure `cron` jobs produce output via e-mail.

FreeBSD 14.2 is now available (https://www.freebsd.org/releases/14.2R/announce/), and as much as it hurts me sometimes, we don't have the same inhibition for using new major releases in this class that we often do in enterprise environments. In business IT, you often want to wait for the first minor version update, let others be early adopters, and sit on the sidelines while they find problems with new releases first.

part zero: get it

Grab the latest FreeBSD 14.2 installation "disc1" ISO from the following URL.

https://download.freebsd.org/ftp/releases/ISO-IMAGES/14.2/

Intel users want the amd64 disc1 ISO, just over a gigabyte in size. ARM Mac users want the arm64 disc1 ISO, just under a gigabyte. Also note, there's an x-zipped version (ending in `.xz`) that'll save you some time downloading, but `xz` is the most successful at shrinking files in part because it's the most CPU-intensive algorithm. Just know that you pay in wait time after you download, if you go this route. I recommend you don't. **You will want to keep this ISO after you're done with lab 2 … you'll use it again in lab 6.**

Look around FreeBSD's website at www.FreeBSD.org … in particular, I'd like you to notice …

… the release notes for version 14.2 (https://www.freebsd.org/releases/14.2R/relnotes/), and within that, the "availability" section explains what the various downloads contain, and in some cases, how to use them. "Release notes" are typical for all software releases … OpenBSD has a very similar document. I just didn't ask you to look in lab 1 … but I suggest you do now.

… the security advisories page (https://www.freebsd.org/security/advisories.html) … click on one of the links. Note how affected versions of each advisory are listed, along with descriptions of the problem, security ramifications, installation directions, and links to signed source code diffs. As we discussed in lab 1, until just recently, all base operating system patching in BSD-land was historically always done manually, with source, by re-compiling affected pieces of the operating system. We'll revert to full binary patching here in lab 2, but still wanted to point this out.

part one: install it

Create a new, **custom** virtual machine in your hypervisor, whichever it is. ARM Mac users, we're using an ARM installation ISO, so choose to virtualize rather than emulate this time. On this system, let's use the following specifications:

- guest OS:
  - Intel/VMware: FreeBSD 14 64-bit
    (if FreeBSD 14 isn't listed in VMware, use version 13 or 12 for now and update VMware)
  - ARM/UTM: virtualize, other
- CPU: two virtual cores/processors
- boot firmware: UEFI
  (this is the default on ARM/UTM)
- RAM: 1024 MB
- hard disk: 16 GB

ARM Mac people, in UTM: change the display card to a "virtio-ramfb" if it's not set to that already, so that you can pause the VM.

Even though FreeBSD is under the "other" menu in VMware products, it has enough market share to get its own entry. Keep in mind, though, as stated on demo day, this selection is just to help VMware choose guest drivers and the best guest hardware, and the "guest OS" configuration option for a slightly older version of FreeBSD should be just fine.

1. Boot the VM, and we can already see from the boot loader menu that we're in for at least a slightly different experience than with OpenBSD … but if we wanted a command line, we could get it here, with option #3. That said, we just want to choose the "boot installer" here.

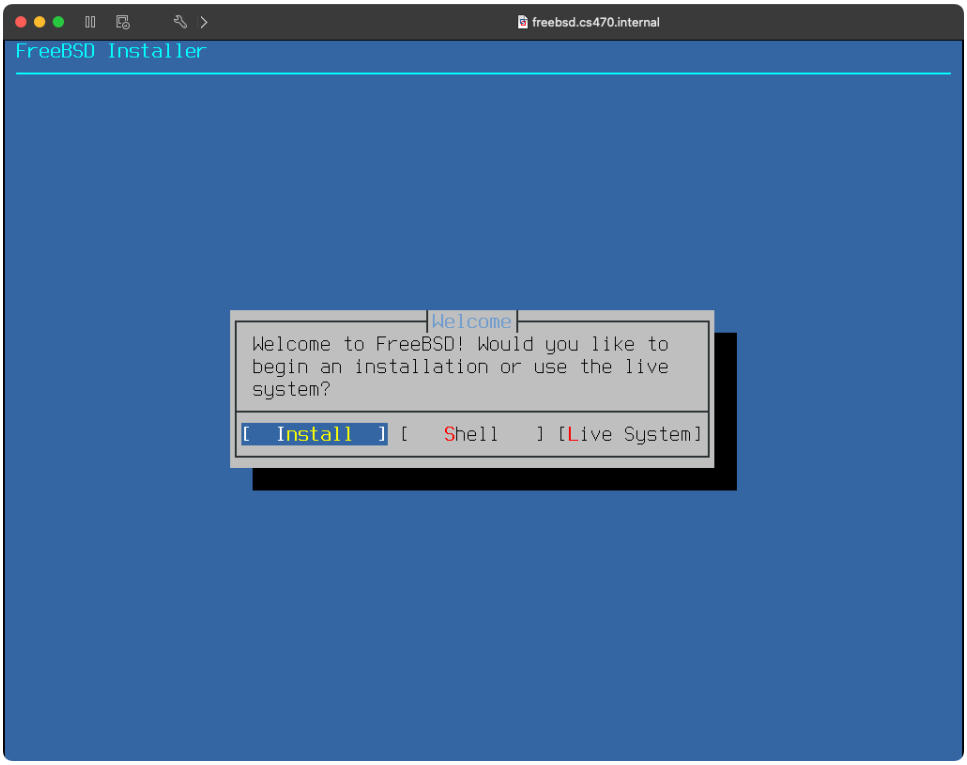
Those of you on Intel are going to see a much snazzier version of the boot loader menu than the older

one on Intel, or those on ARM Macs will.

FreeBSD's mascot is a daemon … not a demon, but a *daemon*. While Beastie (yes, that's his name) looks slightly devilish, he takes his name from BSD itself and from service daemons, and is Unix through and through. His trident is a reference to `fork()`, and he has been visually tweaked over time as the software he represents has evolved. When BSD got the "fast" file system (AKA "FFS" or `ffs`) we discussed in lab 1, Beastie got sneakers in a lot of the BSD merchandising and art, which he's continued to wear to the current day.

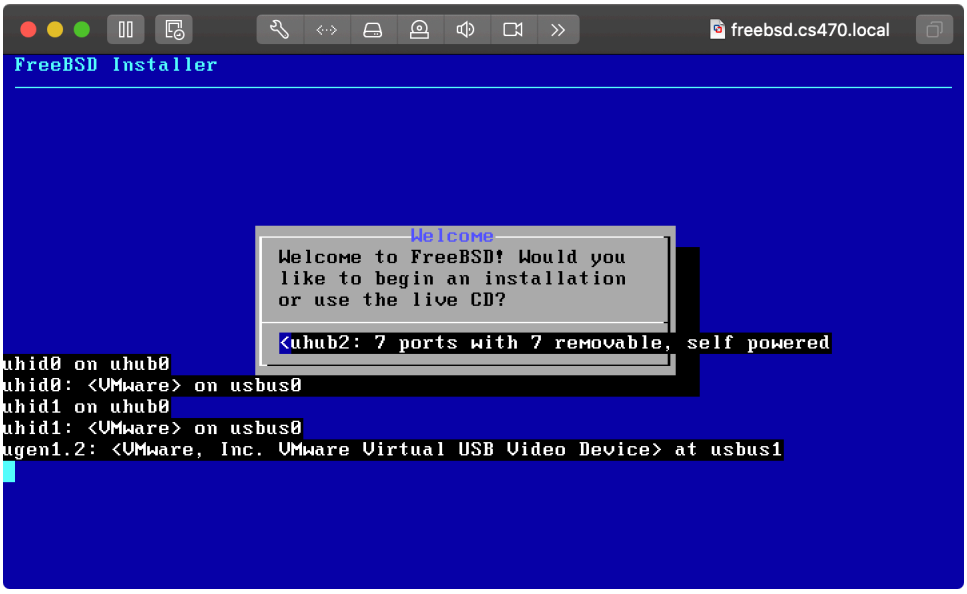
The most popular version of Beastie was drawn by none other than John Lasseter, long before John was the creative leader at Pixar, or the head of Disney Animation.

2. Once you're done with the march of text of the FreeBSD kernel enumerating various devices and which drivers to use with them you are presented with this menu.



In FreeBSD's text-based installer, the tab key changes sub-sections or areas within the FreeBSD installer … generally the arrow keys can be used to move around within areas of an installer "window."

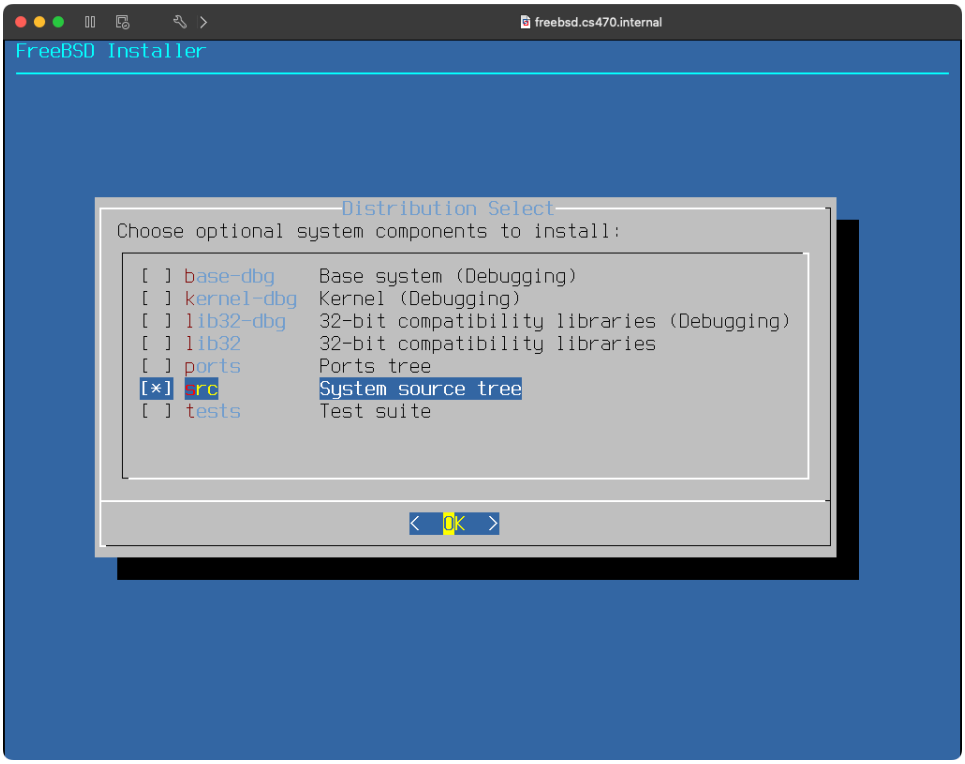
‼ IMPORTANT REMINDER: if you ever get diagnostic messages printed over the installation interface, like in the screenshot below, `control-L` is often honored by many terminals and terminal emulators as a request to re-draw the terminal window. This is an older screenshot from an installation using legacy BIOS I've kept around to show what this looks like, just in case it should also happen to you.



3. You'll see a lot of the options line up similar decisions to those we had to make during the OpenBSD installation process, for instance choosing a keyboard layout ("map"). The default here should be fine. If you mess anything up along the way, you probably won't need to start over; FreeBSD will ask you at the end if you'd like to change anything before rebooting into your freshly-installed VM.

   When asked to set a hostname, use `freebsd` all in lower case.

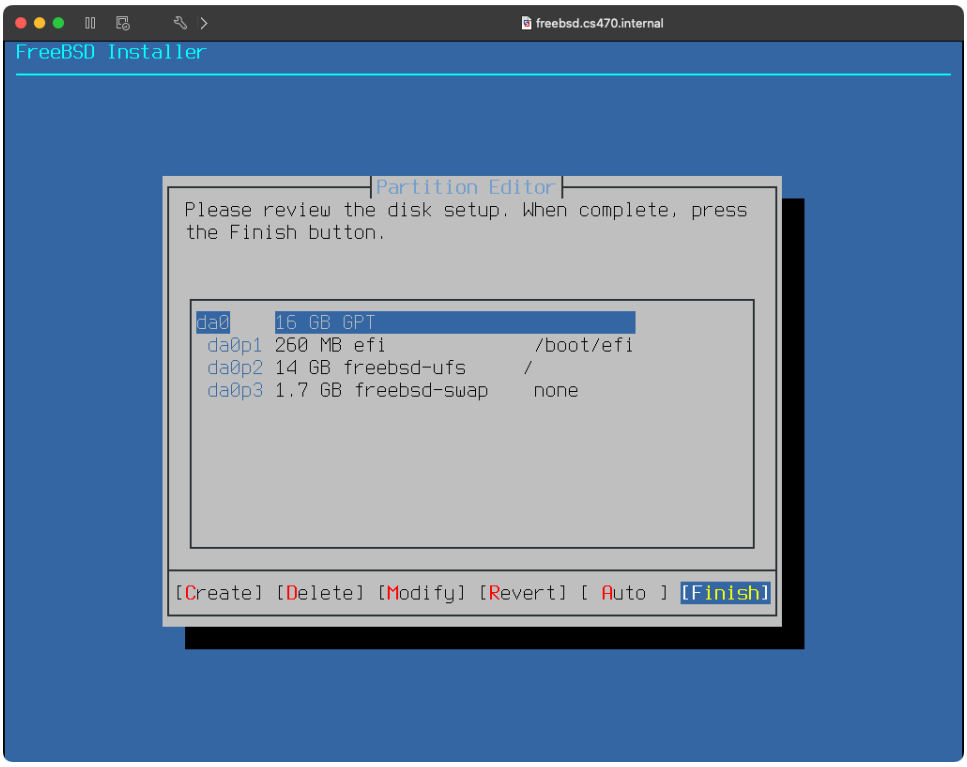
   On the distribution selection page, deselect `kernel-dbg` and `lib32` if they're presented as options, and select `src` … use the space bar here to toggle selections, and in screens like this. Note the `ports` tree here, but don't install it. Just like with OpenBSD and ports-current there, we're going to install a rolling, updated version of the FreeBSD ports tree later.



   At the partitioning dialog, select "Auto (UFS)" and then "entire disk," and GPT as the partitioning scheme. I was pleasantly surprised to find that the automatic configuration in FreeBSD now makes use of a single partition for the file namespace. It should look like the screenshot below, with the bulk of the disk in a single root filesystem. FreeBSD used to use roughly two times the RAM in the VM as a dedicated swap space … now for some reason, it's suggesting about 800 MB of disk. I'm not sure why, but modify your layout to use 14 GB of root filesystem and the remainer (about 1.7 GB) for swap.

   Once upon a time we used to use partitions as the rule rather than the exception; exponentially cheaper disk space is a huge driver for **not** partitioning.

   If your disk device is `ada0`, this is not a problem ... it just means VMware put a virtual ATA or SATA device in your VM (driver name `ada`, the first instance of which is zero, of course). `da` is the device driver name for disks attached to SCSI buses. Inside UTM, FreeBSD uses a `vtbd` (VirtIO) disk device.



All BSD-derived OSs historically used letters `a` through `h` for their partitions. AT&T-derived OSs use the numbers `0` through `7`, as we will see later on in the labs and the lectures. However, unlike OpenBSD with legacy BIOS and MBR, FreeBSD with a GPT partition table no longer uses partition letters. FreeBSD with an MBR partition table still does use partition letters, likely because it has to sub-partition an MBR partition table like we saw OpenBSD do, to support more than four partitions.

After you're done here, select "finish" and the installer will, as all do, ask you to confirm you're really ready to wipe out the prior contents of the drive. You'll then see the "archive extraction" screen … … the `src` tree will take a little while; it is full of **lots** of small files.

You'll then be asked for a root password. Choose a good one!

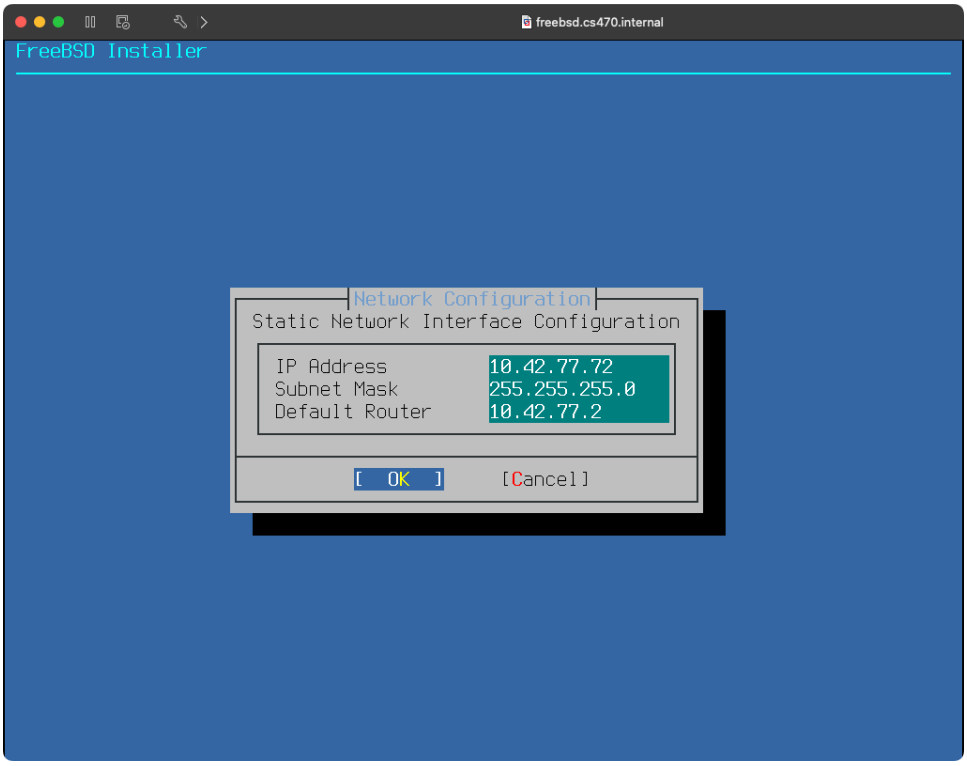
4. Network configuration … you should only have one NIC in your VM. It's been an `le0` device in the past, but the last few times it's been an `em0`. People in UTM will probably have a VirtIO network adapter (`vtnet0`). It doesn't matter which one you have, so long as the installer detects a network interface. FreeBSD, VMware, and UTM all support several kinds of emulated NICs, and just like with the disk devices in the prior step, each set of letters denotes a different driver, in turn for a different set of devices. `em` is an Intel gigabit NIC; `le` is a "Lance" ethernet chip, now put out by AMD.

   We want to configure IPv4 for this interface, but not IPv6. We do NOT want to use DHCP.

   !! IMPORTANT NOTE: if your OpenBSD VM isn't turned up right now, make sure it is, because we're about to be using it as the name server. As a general rule, you will always want ALL your VMs up and

running together during labs. If your laptop starts bogging down, turn down other VMs first, but leave the two BSD-based VMs up! After lab three, the lab three VM will be required as well.

You will want to substitute in the first three octets of your private IP network for what you see in the next screenshot, but everybody's FreeBSD VM should be .72. Of course, use the same router as in lab 1 (on some setups it's .2, on the others it's .1). The default gateway does not change in between computers on a subnet; it is a fixed value for the whole network.



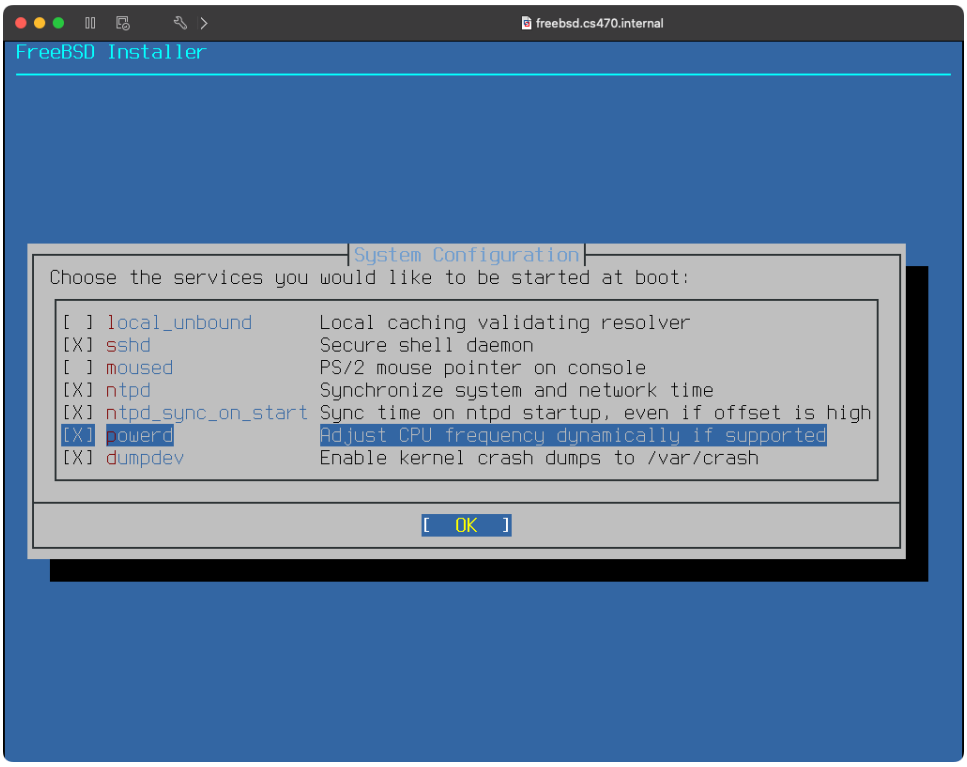
VMware may ask you to provide your password for your "host" (non-virtual) operating system to allow the guest permission to set up the network properly. If it asks, please provide it.

We do **not** want to set up IPv6; again, neither VMware nor UTM support it well, and we'll be using it in none of the labs.

Finally, you'll be asked to provide the resolver configuration for DNS. Your search path should be `cs470.internal` (the DNS namespace we set up in lab 1), and your only name server should be the IP address of your OpenBSD VM.

5. You'll be asked to set the time zone, then to set the date and time.

   In the next screen after that ("system configuration"), you'll be asked to choose services you want to be started at boot. `sshd` and `dumpdev` should be pre-selected, and we want them running. Add `ntpd_sync_on_start` to set the clock at boot time, `ntpd` to keep track of the time as the VM runs, and `powerd`, to save you some electrons.



The "system hardening" screen is next. You may leave all these boxes unchecked.

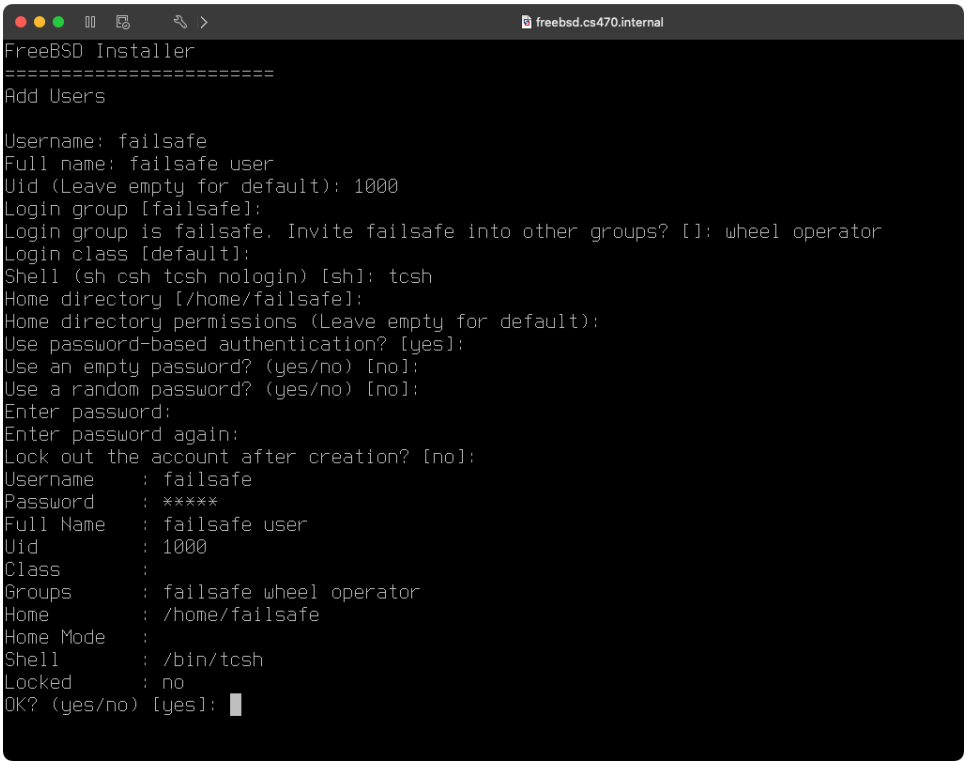
6. You'll then be asked if you want to add users to the installed system … take this opportunity, and do so. We're setting up the `failsafe` user locally, just for use in setting up LDAP and in case it breaks, just as in lab 1. Use user ID number 1000 for `failsafe`, same as the OpenBSD installer used for you. You'll see why we do this later.

   The default login group option (a group of your own) is fine.

   Add your failsafe user to the `wheel` and `operator` groups in the next question, separated by a space, so we can become root with this user (wheel), and shutdown the system and perform backups without becoming root or using `sudo`.

   `bash` doesn't come installed by default with FreeBSD; for the closest thing to it until we get it installed, pick `tcsh` for your shell.

   It should be obvious, but don't lock out the account after creation.



After your user is created, don't create another.

You'll make it to the "final configuration" menu, where you'll have the choice of revisiting a lot of the prior options, if you want to change anything you selected. If you do, great, go! Experiment! If not, or when done, select "exit." Then tell it you don't want a shell, and that you want it to reboot. Your FreeBSD installation is complete, and your system will reboot into multi-user mode.

Make sure you remove the ISO file from the virtual optical drive while it reboots.

7. Log into the console of your FreeBSD system, and make a `.ssh` directory inside `failsafe`'s home folder.

```
$ mkdir -m 700 ~/.ssh
```

Now, copy your SSH public key over into the `authorized_keys` file, as usual, so you can use your key and agent to log into your FreeBSD VM. Test it. Now initial setup is truly complete, and you can use SSH from here on out.

8. Now let's check out where the FreeBSD put most of the options we chose during installation.

```
$ more /etc/rc.conf
```

My output looked like this; of course yours will look slightly different, depending on the first three octets of your VM network and what kind of NIC your VM is using.

```
hostname="freebsd"
ifconfig_em0="inet 10.42.77.72 netmask 255.255.255.0"
defaultrouter="10.42.77.2"
sshd_enable="YES"
ntpd_enable="YES"
ntpd_sync_on_start="YES"
powerd_enable="YES"
moused_nondefault_enable="NO"
# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable
dumpdev="AUTO"
```

Note that the network configuration here in `rc.conf` is mixed with service configurations. Also note, the file is relatively short for a full system configuration. This is because FreeBSD has a set of "default" configuration files under `/etc/defaults` that it reads before local configuration does the rest. Take a second to look briefly through all the settings and on/off switches for built-in services.

```
$ more /etc/defaults/rc.conf
```

This file, by comparison, is much larger … with the goal of giving you a smaller `rc.conf` to look at, where you made only the changes to set or override whatever boot variables you want. On OpenBSD, `/etc/rc.conf.local` (with the changes) is parsed after the defaults in `/etc/rc.conf`.

As we saw in lab 1, OpenBSD stores its network interface configuration(s) in `/etc/hostname.*` files, named by interface driver and number, and its default gateway in `/etc/mygate`. FreeBSD stores its default gateway in a statically-named shell variable (`defaultrouter`), and stores its network interface configuration(s) in shell variables, named by device name and number.

9. Remember the login banner? Fear not if you don't … since the dawn of time it's been in `/etc/motd` like OpenBSD, and will be found in the same place on virtually every OS we play with. FreeBSD's `motd` is now dynamically generated, a change that seems to have been driven by jealousy of the dashboard-like information presented during a login into Ubuntu, which we'll see soon, in lab 4. Presumably, we'll be able to add more that isn't there quite yet ... the template and the target seem static, and quite identical.

```
$ more /etc/motd.template

Welcome to FreeBSD!

Release Notes, Errata: https://www.FreeBSD.org/releases/
Security Advisories:   https://www.FreeBSD.org/security/
FreeBSD Handbook:      https://www.FreeBSD.org/handbook/
FreeBSD FAQ:           https://www.FreeBSD.org/faq/
Questions List: https://lists.FreeBSD.org/mailman/listinfo/freebsd-questions/
FreeBSD Forums:        https://forums.FreeBSD.org/

Documents installed with the system are in the /usr/local/share/doc/freebsd/
directory, or can be installed later with:  pkg install en-freebsd-doc
For other languages, replace "en" with a language code like de or fr.

Show the version of FreeBSD installed:  freebsd-version ; uname -a
Please include that output and any error messages when posting questions.
Introduction to manual pages:  man man
```

```
FreeBSD directory layout:      man hier

To change this login announcement, see motd(5).
```

The FreeBSD Handbook is a far more comprehensive walkthrough-style documentation for a near-complete set of features of FreeBSD. We'll be talking shortly about how various operating system file hierarchies are laid out, but I wanted to make a point of showing this off … most other operating systems don't have a neat `man` page with a guided tour through their file tree.

```
$ man hier
```

Again, use `return` to scroll through line-by-line, space to load another page.

If you need to shut down your FreeBSD VM, FreeBSD supports the same syntax for `shutdown` as OpenBSD does. FreeBSD also implements `poweroff`, which OpenBSD doesn't have. This tool came from AT&T Unix, and has been included with most Linux distributions for some time. It comes to BSD-land last, of course, finally having won acceptance after a long time in the limbo of Linux-land. `poweroff` is effectively an alias for `shutdown -p now`.

part two: installing third-party software, both from source and package

Now let's install some software on your FreeBSD VM. Since we used the ports tree to build from source in OpenBSD and lab 1, and since I initially faced some issues with the ports tree on my ARM Mac, we'll be exploring the binary package manager (`pkg`) in this section of the lab.

The package manager helps manage things installed via the ports tree as well, because in both OpenBSD and FreeBSD, when you build a port, you build the package, and then install the software from that package. In fact, the ports tree is how the operating system team automates building the third-party software binaries it distributes via the package manager. As such, the software installed by both methods are effectively interchangeable, and you can manage/upgrade/uninstall software installed both ways by using `pkg` on both operating systems.

For many years, FreeBSD had a utility called `portsnap` for managing the ports tree and keeping it up to date with differential updates, in order to minimize the traffic required to host the ports tree. During the lifecycle of FreeBSD 13, the development team completed a transition to using `git` for source code control for the entire operating system. `git` comes from the Linux kernel project, we'll play with it in lab 7, and it is what most source code repositories are going to for version control. In FreeBSD 14, `portsnap` has been removed, and we use `git` to download the ports tree … so there's no way around using `pkg` briefly, at least to bootstrap the ports tree if we want to track a current copy of it, with security fixes and updates.

10. Let's install and initialize FreeBSD's pre-compiled binary package manager … `pkg` moves so quickly that it used to be shipped as a part of FreeBSD but updated after virtually every installation. Now they've given up on shipping `pkg` at all.

    Note that you find a tool called `pkg` in `/usr/sbin` …

```
$ which pkg
/usr/sbin/pkg
```

… but this just exists to tell you that `pkg` really isn't there yet, and has to be installed, as you'll see. Use `su` to become root (note the pound sign prompt below) and run the following command.

```
# pkg update
```

… key in a `y` at the right time to override the default answer and install `pkg`.

```
The package management tool is not yet installed on your system.
Do you want to fetch and install it now? [y/N]: y
Bootstrapping pkg from pkg+https://pkg.FreeBSD.org/FreeBSD:14:amd64/quarterly,
please wait...
Verifying signature with trusted certificate pkg.freebsd.org.2013102301... done
Installing pkg-1.21.3...
Extracting pkg-1.21.3: 100%
Updating FreeBSD repository catalogue...
Fetching meta.conf: 100%    178 B   0.2kB/s    00:01
Fetching data.pkg: 100%    7 MiB   7.3MB/s    00:01
Processing entries: 100%
FreeBSD repository update completed. 35854 packages processed.
All repositories are up to date.
```

If the above command failed because of a "host not found" error then your DNS server or network setup is somehow messed up. Did you get your name server working properly and validated? This is the first time you're really using it outside your OpenBSD VM.

Now that we actually have `pkg` installed instead of just the placeholder, you can run `pkg` with its `info` verb to show the list of installed ports and packages.

```
$ pkg info
pkg-1.21.3                     Package manager
```

Not much, but it's a start. Just like on OpenBSD, FreeBSD installs third-party software and software under `/usr/local` to keep it separate, so local versions of tools can be prioritized over built-in operating system defaults.

11. A copy of the ports tree was offered to us during installation, but we chose not to accept. That version of the ports tree was a "release" of the ports tree, with no versioning data included, and thus no real capability to update it. We're going to want to update it, so we're going to use a rolling version of the ports tree.

    OpenBSD uses a source code versioning system, `cvs`, to check out and keep a copy of OpenBSD's ports tree up to date. As we mentioned before, FreeBSD used to have its own tool for keeping a rolling copy of the ports tree up to date, but now FreeBSD uses version control as well. The ports tree is little more than a large tree of shell scripts, so using source code control tools for this is the right tool for the job. FreeBSD has leapt ahead of OpenBSD by using the modern `git` source control tool instead of the antiquated `cvs`, which was designed in 1986.

Use `su` to become root again if needed, and install `git`.

```
# pkg install git
```

Note that FreeBSD updates its repository "catalogue," the local cache of available packages, with each invocation, and that dependency checking is done for us. I've trimmed the following output to cut out all the dependencies; don't worry if your output has minor differences.

```
Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
All repositories are up to date.
Updating database digests format: 100%
The following 37 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
      …
      git: 2.48.1
      …

Number of packages to be installed: 35

The process will require 326 MiB more space.
56 MiB to be downloaded.

Proceed with this action? [y/N]:
```

After you key in another `y` to confirm the operation, you'll see it first download ("fetch") the packages, then extract and install each of them. After the lot of them is done, you'll see the following.

```
[37/37] Installing git-2.48.1...
===> Creating groups.
Creating group 'git_daemon' with gid '964'.
===> Creating users
Creating user 'git_daemon' with uid '964'.
[37/37] Extracting git-2.48.1: 100%
=====
Message from python311-3.11.11:

--
Note that some standard Python modules are provided as separate ports
as they require additional dependencies. They are available as:

py311-gdbm       databases/py-gdbm@py311
py311-sqlite3    databases/py-sqlite3@py311
py311-tkinter    x11-toolkits/py-tkinter@py311
=====
Message from git-2.48.1:

--
If you installed the GITWEB option please follow these instructions:

In the directory /usr/local/share/examples/git/gitweb you can find all files to
make gitweb work as a public repository on the web.

All you have to do to make gitweb work is:
1) Please be sure you're able to execute CGI scripts in
   /usr/local/share/examples/git/gitweb.
2) Set the GITWEB_CONFIG variable in your webserver's config to
   /usr/local/etc/git/gitweb.conf. This variable is passed to gitweb.cgi.
3) Restart server.
```

```
If you installed the CONTRIB option please note that the scripts are
installed in /usr/local/share/git-core/contrib. Some of them require
other ports to be installed (perl, python, etc), which you may need to
install manually.
```

Just like when installing third-party software in OpenBSD, with either the ports tree or the package manager, in FreeBSD the port maintainer can also give you a heads up after the end of the installation process. This time, none of the messages left behind are pertinent to anything we'll be doing – we are neither using the referenced `py311-*` modules nor are we setting up a `git` server – but always be vigilant and read what your system says to you. Also, note that `git` came along with its own service account and group.

The above output is provided for context; you will start to see the paths of ports listed, and this whole process will take a few minutes. Not a really long time … just a few minutes. We're grabbing a very large set of small files from online sources.

12. Now, we install the ports tree.

```
# git clone --depth=1 https://git.FreeBSD.org/ports.git /usr/ports
```

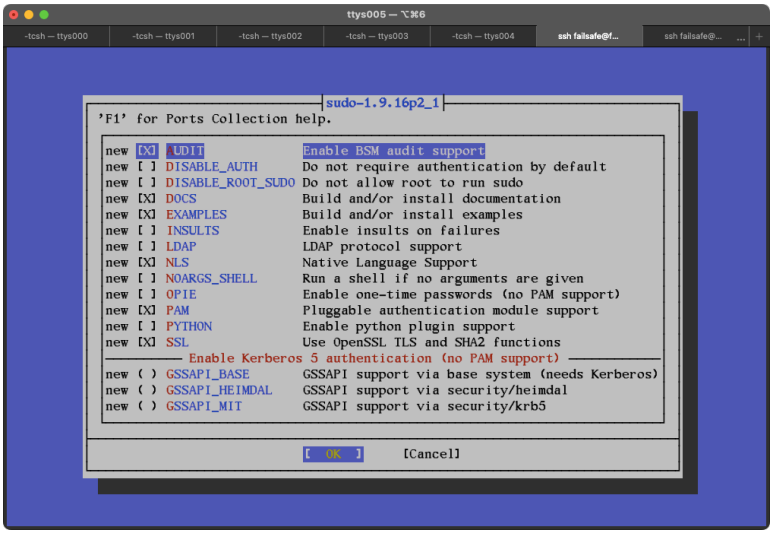
This would normally take a while, on the order of an hour or so, even with a fast internet connection. However, since we don't care about the commit history and only want the latest state of the repository, we specify `--depth=1` and `git` only grabs the most recent state of the main branch. With `git`, we also get brief output showing us how far along the operation is, and how fast it's moving. `cvs` and `portsnap` just gave us a long list of files scrolling by, and no real idea of how far along we were in the process. Unfortunately, there's a long delay at the end of the process, after "checking objects." **Let it run until you get a command line back.**

13. Now let's install `sudo`, so that you don't have to walk around as root. Run the following command … for the last time here, I'm going to remind you the pound sign prompt means that you ran `su` to become root, like in the prior step, then you type the command after it … don't type the `$` or the `#` …

```
# cd /usr/ports/security/sudo && make install
```

… and yes, I want you to use the ports tree and compiler for this build, not the package manager, for the purpose of demonstrating some differences between FreeBSD and OpenBSD.

You will notice FreeBSD building pre-requisites first, and at a certain point during the installation, you'll see a menu like this …



… where FreeBSD has selectable features for software in the ports tree, it will present you this screen to allow you to choose the options. Some of you have noted OpenBSD's by-default inclusion of the "insults" option … this is the code that pokes fun at you when you fat-finger your password. I have NOT chosen LDAP here amongst the features here, because `sudo` worked just fine without making it aware of LDAP on OpenBSD.

Select "OK" when you're done looking around here.

You may still be asked to select options for a couple dependencies after that, so don't walk away for too long … come back soon. This can be dealt with in future ports compiles by defining the macro BATCH on the command line (as in `make -DBATCH install`).

‼ IMPORTANT REMINDER: when an error happens while building ports, scroll upward to find the **very first** error message. The last is often the effect of a resulting cause, or a resulting cause of a resulting cause, and we want to know where the problems began, the "root cause." This is why we often send output to a file, and prefer to SSH into our VM and use software-based terminals instead of the console … it makes it way, way easier to scroll back to find the root cause of a problem.

If you get the error `Invalid configuration `arm64-portbld-freebsd14.2': machine `arm64-portbld' not recognized` … please let me know. For some reason, I seem to be the only one ever affected. **If and only if you get this build error**, also check out this page.

You can now type `exit` to quit your root shell (that you started with `su`). Now that we have `sudo` installed, you don't need to be root all the time, just to put `sudo` in front of commands you want to be run as root.

‼ IMPORTANT NOTE: typos with `sudo` can be just as painful as typos on a root command line. ALWAYS double check commands to be run with elevated privileges before hitting the return key.

14. Next, let's install `bash`. From here on out, if you'd rather use `pkg` to get quicker results for adding software, be my guest. If you want to install from the binary package …

    ```
    $ sudo pkg install bash
    ```

    … if you want to build from source …

    ```
    $ cd /usr/ports/shells/bash && sudo make install
    ```

    `sudo` will ask you for your password, and again, this is your user account password, not the root password you configured at installation. Unless you did something different from me, you're presented with the following error message:

    ```
    failsafe is not in the sudoers file.
    This incident has been reported to the administrator.
    ```

    Oops, looks like we need to set up `sudo` first. Become root again, with `su`, and run …

    ```
    # visudo
    ```

    … note how FreeBSD puts configuration files that would ordinarily go in `/etc` into `/usr/local/etc` for software installed from ports and packages. While you're editing the `sudoers` file in `vi`, uncomment the line allowing the `wheel` group to run any command, by removing the pound sign at the beginning … **but not the line allowing them to do so without a password.**

    Note that the `sudoers` file is treated with the utmost caution … the permissions on this file cause it to be opened in read-only mode in `vi` …

    ```
    root@freebsd:/home/failsafe # ls -l /usr/local/etc/sudoers
    -r--r-----  1 root wheel 4747 Jun 11 01:09 /usr/local/etc/sudoers
    ```

    … so you will have to use a bang (`!`) after `:wq` to tell `vi` that you REALLY want to write out the file and bypass its read-only permissions, even as root.

15. Now that we've fixed `sudo`, let's try to install `bash` again.

    ```
    …
    The following 1 package(s) will be affected (of 0 checked):

    New packages to be INSTALLED:
          bash: 5.2.37

    Number of packages to be installed: 1

    The process will require 9 MiB more space.
    2 MiB to be downloaded.

    Proceed with this action? [y/N]: y
    [1/1] Fetching bash-5.2.37.pkg: 100%    2 MiB   1.8MB/s    00:01
    Checking integrity... done (0 conflicting)
    [1/1] Installing bash-5.2.37...
    [1/1] Extracting bash-5.2.37: 100%
    ```

Piece of cake. Now we have `bash`, and like in OpenBSD, you can use the `chsh` command to change your shell to `/usr/local/bin/bash`. Note that `chsh` drops you into a `vi` window where you're expected to alter a temporary file, which gets parsed and read back into `/etc/passwd` and `/etc/master.passwd`, like in OpenBSD.

When the ports tree or package manager installs a shell in FreeBSD, it's added to the file `/etc/shells`, which contains the list of acceptable values for a user's login shell in `/etc/passwd`. Since `bash` has been installed to `/usr/local/bin`, it matches what we already have in LDAP.

16. Next, let's add GnuPG. Again, we need this on every VM for the grading software later in the class.

```
$ sudo pkg install gnupg
```

Please feel free to install from package or build from source using the ports tree … again, they are interchangeable.

```
$ cd /usr/ports/security/gnupg && sudo make -DBATCH install
```

GnuPG has a lot of dependencies, so once it's started running, this might be a good time to grab something to drink, stretch, or do whatever you do if you chose to give your computer a workout and compile it from source.

17. Let's install the root CA certificates package, `ca_root_nss`. Since this is just a package of certificate files, there's not much to compile … just use `pkg`.

```
$ sudo pkg install ca_root_nss

Scanning /usr/share/certs/untrusted for certificates …
Scanning /usr/share/certs/trusted for certificates …
Scanning /usr/local/share/certs for certificates …
…
FreeBSD does not, and can not warrant that the certification authorities
whose certificates are included in this package have in any way been
audited for trustworthiness or RFC 3647 compliance.

Assessment and verification of trust is the complete responsibility of
the system administrator.

This package installs symlinks to support root certificate discovery
for software that either uses other cryptographic libraries than
OpenSSL, or use OpenSSL but do not follow recommended practice.

If you prefer to do this manually, replace the following symlinks with
either an empty file or your site-local certificate bundle.

  * /etc/ssl/cert.pem
  * /usr/local/etc/ssl/cert.pem
  * /usr/local/openssl/cert.pem
```

Since we've already set up a root CA in lab 1b, you should be able to understand this message and what it's telling you about trusting other CA certificates. If you don't, read up and make sure you do. Of course, we're going to be touching the files this message references.

18. Now, let's install some mail server software. A piece of software for getting mail to and in between mail servers (that is, implementing the SMTP protocol), `sendmail`, is already included with FreeBSD. We just need software to implement mail folders for, and accept delivery of, mail messages. Let's install `dovecot`.

    ```
    $ sudo pkg install dovecot
    ```

    If you'd like to build from source instead …

    ```
    $ cd /usr/ports/mail/dovecot && sudo make install
    ```

    You should take notice of feedback from ports and packages when they install on FreeBSD. The output of installation payloads often includes pertinent information for running the software correctly, more securely, and for getting it running in the first place. As you can see, all these are included in the message from `dovecot`.

    …

    ```
    [3/3] Installing dovecot-2.3.21.1_1...
    ===> Creating groups.
    Creating group 'dovecot' with gid '143'.
    Creating group 'dovenull' with gid '144'.
    ===> Creating users
    Creating user 'dovecot' with uid '143'.
    Creating user 'dovenull' with uid '144'.
    [3/3] Extracting dovecot-2.3.21.1_1: 100%
    --
    Message from dovecot-2.3.21.1_1:

    --
    You must create the configuration files yourself. Copy them over
     to /usr/local/etc/dovecot and edit them as desired:

          cp -R /usr/local/etc/dovecot/example-config/* \
                /usr/local/etc/dovecot

     The default configuration includes IMAP and POP3 services, will
     authenticate users agains the system's passwd file, and will use
     the default /var/mail/$USER mbox files.

     Next, enable dovecot in /etc/rc.conf:

          dovecot_enable="YES"
    …
    ```

    `dovecot` is installed. There are some warnings that followed this message; I'm not putting them here because I'm telling you about them here and now and I don't anticipate that we'll step on them, but you should read them anyways, because it's a good practice, and just in case one of these settings looks tempting later for some reason in your exploratation with `dovecot`, Very similarly to how we turned on services in OpenBSD, we'll enable it as suggested in `/etc/rc.conf` shortly … but first we need to set up a user to receive mail.

    Speaking of users, the prior step also installed the OpenLDAP client package for me. If it didn't also for you, we'll be dealing with that in a couple of steps.

part three: configuring LDAP login

Now let's get our new FreeBSD instance plugged into LDAP, so we can make use of all our users and groups.

!! THE WARNINGS FROM LAB 1B ABOUT LOCKING YOURSELF OUT OF YOUR VM … THEY APPLY AGAIN HERE. You may want to keep a root shell open in a terminal window throughout this section, until you're sure it works. You may have to go back into single-user mode if you really mess up while configuring authentication.

Single-user mode works the same here in FreeBSD as in OpenBSD; you will have to remount your root filesystem read/write to change anything and set up your terminal to use `more` or `vi` … but you should know what to do, and you've been warned now.

19. Before we perform authentication over a network connection we **always** need to provide for encryption. The LDAP software we're about to set up here is a client, and doesn't need its own certificate, but it does need to know that it trusts the LDAP server's certificate.

    Just like OpenBSD, FreeBSD stores a pile of CA certificates at `/etc/ssl/cert.pem` … and just like we did on OpenBSD, we're going to add our new CA certificate to it.

    ```
    $ scp -p openbsd:/home/pki/data/cacert.pem ~failsafe
    ```

    Note that because you are failsafe on this computer, the SSH client tries to log you in as failsafe there, unless you tell it otherwise.

    Please also note what the SSH client is telling you here. Because we did not use something like the CA to sign our OpenBSD VM's SSH server key pair, it's offering us a key fingerprint (really, a SHA256 hash) that we can use to go make sure we're really talking to the computer we want to talk to, and provide *some* (not perfect) assurance that nobody is eavesdropping our network traffic.

    ```
    The authenticity of host 'openbsd.cs470.internal (10.42.77.71)' can't be
    established.
    ED25519 key fingerprint is SHA256:3/KnJPOxA4Q2cp3O2A84MT5onc0fvNcwuxBpDQwmLo0.
    This key is not known by any other names.
    Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
    Warning: Permanently added 'openbsd.cs470.internal' (ED25519) to the list of known
    hosts.
    cacert.pem                                    100% 7091     1.5MB/s   00:00
    ```

    The presumption made here is that a user of any given system has another way of validating the key, that it's worth the time involved in doing so. Users generally **DO** have the ability to read the public pieces of SSH server configuration in `/etc/ssh` … take EDORAS as proof, this is just the default configuration of those files, and it makes sense for it to be this way.

    ```
    $ ssh edoras.sdsu.edu ls -l /etc/ssh
    total 1
    -rw-r--r--  1 root root     577388 Aug 14  2024 moduli
    -rw-r--r--  1 root root       1770 Aug 14  2024 ssh_config
    drwxr-xr-x  2 root root         27 Jan  6 02:31 ssh_config.d
    -rw-------  1 root root       4741 Sep 18 09:39 sshd_config
    -rw-------  1 root root       4404 Jan  3  2017 sshd_config.R
    -rw-------  1 root root       4315 Aug 14  2024 sshd_config.rpmnew
    ```

```
-rw-r-----. 1 root ssh_keys    227 Mar 17  2016 ssh_host_ecdsa_key
-rw-r--r--. 1 root root        162 Mar 17  2016 ssh_host_ecdsa_key.pub
-rw-r-----. 1 root ssh_keys    387 Mar 17  2016 ssh_host_ed25519_key
-rw-r--r--. 1 root root         82 Mar 17  2016 ssh_host_ed25519_key.pub
-rw-r-----. 1 root ssh_keys   1679 Mar 17  2016 ssh_host_rsa_key
-rw-r--r--. 1 root root        382 Mar 17  2016 ssh_host_rsa_key.pub
```

This means that all you have to do is log into that other computer, for which you have presumably already validated the encryption key. **On your OpenBSD VM**, generate the hash from the public key file in the SSH server configuration file. Ordinarily, you should do this before answering `yes` when SSH asks if you still want to connect.

```
$ ssh-keygen -lf /etc/ssh/ssh_host_ed25519_key.pub
256 SHA256:3/KnJPOxA4Q2cp3O2A84MT5onc0fvNcwuxBpDQwmLo0 root@openbsd.localdomain
(ED25519)
```

Because the two hashes match, you know it's the right key. This provides *some* assurance that the traffic between the two isn't being eavesdropped. Sometimes things happen – like installing an SSH software upgrade or re-installing a computer's operating system – that cause the primary SSH key to change. Once we accept an SSH key for a system, we only hear about it when it changes.

We have to go through this pain if we want to validate keys with SSH, because we're not using certificates typically, out of the box with SSH. SSH host keys aren't signed by a trusted third party, so there is no chain of trust to validate keys with … and this is why we go through the additional pain of setting up certificates with other services. Certificates **can** be used for both host keys and authentication with SSH, but I've rarely seen it set up. This system of manual validations seems to be considered good enough for most, apparently.

Back to the CA cert task … **on your FreeBSD** VM, let's look in `/etc/ssl` ...

```
lrwxr-xr-x  1 root wheel    43 Feb 21 17:10 cert.pem ->
../../usr/local/share/certs/ca-root-nss.crt
drwxr-xr-x  2 root wheel  3072 Feb 26 13:14 certs
-rw-r--r--  1 root wheel 12336 Nov 29 02:21 openssl.cnf
drwxr-xr-x  2 root wheel  1536 Feb 26 13:14 untrusted
```

… as you can see it's got a `cert.pem` file, but FreeBSD also has a directory full of certificates here in `/etc/ssl/certs` that will be more resilient to OS updates, so we're going to drop our certificate in there …

```
$ sudo cp -p ~failsafe/cacert.pem /etc/ssl/certs/CS470.crt
```

According to convention, this directory also wants a symbolic link to the certificate, named for a short hash/fingerprint of the certificate …

```
$ openssl x509 -noout -hash -in ~failsafe/cacert.pem
8550b8fe
```

… let's do this the Unix way, instead of copying and pasting …

```
$ sudo ln -s CS470.crt /etc/ssl/certs/`openssl x509 -noout -hash -in
```

```
~failsafe/cacert.pem`.0
```

… note that there are no spaces before and and after the backticks in the nested command … we want it to be a part of the absolute path we're referring to under `/etc/ssl/certs`.

```
$ ls -l /etc/ssl/certs/ | grep CS470
lrwxr-xr-x  1 root     wheel       9 Feb 26 16:19 8550b8fe.0 -> CS470.crt
-rw-rw-r--  1 failsafe failsafe 7091 Feb  8 12:51 CS470.crt
```

You should now be able to verify the certificate on your OpenBSD VM's LDAP server via OpenSSL, and with all software that uses OpenSSL on your FreeBSD VM:

```
$ openssl s_client -connect openbsd.cs470.internal:636 | grep -i -e verify
depth=1 C = US, ST = Californistan, O = SDSU CS470, CN = CS470 2025 Spring CA
verify return:1
depth=0 C = US, ST = Californistan, O = SDSU CS470, CN = openbsd.cs470.internal
verify return:1
Verify return code: 0 (ok)
```

Let's also save the certificate file in `/etc/ssl` for safe keeping. This way, if we ever need to rebuild the cert trust database, we'll be happy to have a copy of the only custom certificate right there …

```
$ sudo mv ~failsafe/cacert.pem /etc/ssl/
$ sudo chown root:wheel /etc/ssl/cacert.pem
```

20. In what should be an unsurprising next move, we're going to install OpenLDAP, specifically, the client for OpenLDAP version 2.6, if it wasn't already installed along with dovecot a couple of steps ago. Choose your weapon, either building from source …

```
$ cd /usr/ports/net/openldap26-client && sudo make -DBATCH install
```

… or installing from the package …

```
$ sudo pkg install openldap26-client
```

… either way, you should see something like the following …

```
Message from openldap26-client-2.6.9:

--
The OpenLDAP client package has been successfully installed.

Edit
  /usr/local/etc/openldap/ldap.conf
to change the system-wide client defaults.

Try `man ldap.conf' and visit the OpenLDAP FAQ-O-Matic at
  http://www.OpenLDAP.org/faq/index.cgi?file=3
for more information.
```

… or something like this if OpenLDAP's installation was taken care of previously.

```
The most recent versions of packages are already installed
```

As you might imagine, we're going to be taking OpenLDAP up on its invitation shortly.

21. FreeBSD also requires two more ports to be installed in order to authenticate using an LDAP directory, `pam_ldap`, the PAM (pluggable authentication modules) support for LDAP, and `nss_ldap`, name service switch (user and group mapping and more) support for LDAP. Either from source …

```
$ cd /usr/ports/security/pam_ldap && sudo make install
```

… or …

```
$ sudo pkg install pam_ldap
…

===>  Installing for pam_ldap-186_2
===>  Checking if pam_ldap is already installed
===>   Registering installation for pam_ldap-186_2
Installing pam_ldap-186_2...
Edit /usr/local/etc/ldap.conf in order to use this module.  Then
create a /usr/local/etc/pam.d/ldap with a line similar to the following:

login auth  sufficient  /usr/local/lib/pam_ldap.so
```

… now, the other, from source …

```
$ cd /usr/ports/net/nss_ldap && sudo make install
```

… or from package …

```
$ sudo pkg install nss_ldap
```

I'm leaving this here, just in case you should run into this bug, but I doubt you will. **Read this**, but unless you hit the bug somehow, there's nothing else for you to do in this step but read.

```
cc -DHAVE_CONFIG_H -I.   -I/usr/local/include -DLDAP_REFERRALS -DLDAP_DEPRECATED -
DPIC -D_REENTRANT  -O2 -pipe  -DHAVE_LOGIN_CLASSES -DLDAP_DEPRECATED -fstack-
protector-strong -fno-strict-aliasing  -Wall -fPIC -MT ldap-init-krb5-cache.o -MD -
MP -MF .deps/ldap-init-krb5-cache.Tpo -c -o ldap-init-krb5-cache.o ldap-init-krb5-
cache.c
ldap-init-krb5-cache.c:102:10: fatal error: 'profile.h' file not found
#include <profile.h>
         ^~~~~~~~~~~
1 error generated.
gmake[3]: *** [Makefile:501: ldap-init-krb5-cache.o] Error 1
gmake[3]: Leaving directory '/usr/ports/net/nss_ldap/work/nss_ldap-265'
gmake[2]: *** [Makefile:396: all] Error 2
gmake[2]: Leaving directory '/usr/ports/net/nss_ldap/work/nss_ldap-265'
===> Compilation failed unexpectedly.
Try to set MAKE_JOBS_UNSAFE=yes and rebuild before reporting the failure to
the maintainer.
*** Error code 1
```

This error appeared over the summer of 2023, first for some students who took a while to get to lab 2, because it wasn't initially there when I initially updated this lab for that summer class, and we had to work around this through the fall semester. This happened because one of the FreeBSD port maintainers was modifying this port's configuration scripts to better detect FreeBSD and add support

for Kerberos. You can't see it now, because `git` is stepping on file timestamps, but could see the offending change then by looking in the `files` subdirectory of the port directory …

```
$ ls -lt files
total 1
-rw-r--r--  1 root  wheel  1593 May  9 22:22 patch-configure.in
-rw-r--r--  1 root  wheel   236 Aug 14  2019 pkg-message.in
-rw-r--r--  1 root  wheel  9655 Aug 13  2018 bsdnss.c
-rw-r--r--  1 root  wheel   791 Aug 13  2018 patch-ldap-netgrp.c
-rw-r--r--  1 root  wheel  1027 Jul 23  2018 patch-Makefile.am
-rw-r--r--  1 root  wheel   875 Jul 23  2018 patch-ldap-ethers.c
-rw-r--r--  1 root  wheel   179 Jul 23  2018 patch-ldap-ethers.h
-rw-r--r--  1 root  wheel   648 Jul 23  2018 patch-ldap-pwd.c
-rw-r--r--  1 root  wheel  1371 Jul 23  2018 patch-login-classes
-rw-r--r--  1 root  wheel   292 Nov 13  2014 patch-ldap-grp.c
-rw-r--r--  1 root  wheel  1885 Nov 13  2014 patch-ldap-init-krb5-cache.c
-rw-r--r--  1 root  wheel   661 Jan 14  2014 patch-ldap-nss.c
-rw-r--r--  1 root  wheel   415 Jan 14  2014 patch-ldap-nss.h
```

… note that the `t` switch makes `ls` sort by time, and put the offending file (`patch-configure.in`) right up top.

Note that on tracking ticket for this bug (https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=270465), enterprising former CS 470 student Anna did a great job finding this page while doing her research. Two workarounds are proposed here: to undo the offending commit, by Dutchman01, and the other, by Einar Bjarni Halldórsson, is to turn off Heimdal (one of the two major distributions of Kerberos) while building `nss_ldap`. This is the cleaner of the two fixes – it doesn't modify the port files – and allowed the package to build successfully.

First, the problem in the `Makefile` by manually disabling Kerberos (Heimdal).

```
$ sudo vi work/nss_ldap-265/Makefile
```

Add `-DHEIMDAL=off` to the end of the line defining the variable `CPPFLAGS`.

```
$ cd work/nss_ldap-265 && sudo make clean && cd ../.. && sudo make install
```

We have to go down into the work directory to run `make clean`, because if we tell the port's `Makefile` to `make clean`, it'll delete the whole work directory, including the fix.

If you run into this bug, please let me know ASAP.

After you've installed it successfully, you should see the below. Note that 1.265_14 was the broken version of the port … the underscore separates the FreeBSD port version number from the author's version number.

```
Message from nss_ldap-1.265_15:

--
The nss_ldap module expects to find its configuration files at the
following paths:

LDAP configuration:     /usr/local/etc/nss_ldap.conf
```

```
LDAP secret (optional): /usr/local/etc/nss_ldap.secret
```

How helpful that each advertised where its configuration files are located, no? We'll take them both up on that as well, in turn. Also note that FreeBSD stores configurations for third-party software installed by the ports tree under `/usr/local/etc` whereas OpenBSD puts both OS-bundled and post-installed software configuration in `/etc`.

22. Configure the LDAP client utilities and libraries in `/usr/local/etc/openldap/ldap.conf` … they both look at the same configuration file, and you've seen this OpenLDAP configuration file before (under `/etc` as we just discussed) in lab 1b. You should know what to do with this file. Do it.

    Make sure you aim `TLS_CACERT` at one of the instances of our root CA certificate that we installed under `/etc/ssl` a couple steps ago.

23. Configure PAM LDAP login support. This starts in `/usr/local/etc/ldap.conf`.

    Set `host` to the fully-qualified DNS domain name of your LDAP server, your OpenBSD VM.

    Set the `base` parameter appropriately for our domain.

    Add a new `uri` line with the following contents:
    ```
    uri ldaps://openbsd.cs470.internal/
    ```

    As FreeBSD's own documentation (https://docs.freebsd.org/en/articles/ldap-auth/#client) advises, uncomment the `pam_login_attribute` line to explicitly search for users in the directory where the `uid` attribute in the directory is the username used in a login attempt.

    Finally, make sure to uncomment either `tls_cacertfile` or `tls_cacertdir` and make sure it's properly aimed at the correct root CA cert file or trust anchor directory.

24. Let's configure PAM itself. Since the first letter in PAM stands for "pluggable" you'll be seeing this again in Linux, and you'll be configuring LDAP next to other modular authentication options.

    Please insert this line …

    ```
    auth            sufficient      /usr/local/lib/pam_ldap.so      no_warn
    ```

    … under the authentication line with `pam_ssh.so` in `/etc/pam.d/imap`, `/etc/pam.d/sshd`, and `/etc/pam.d/system`.

25. The "name service switch" is a piece of software a lot of Unix systems, Solaris included, have used to govern swapping out replacements for the built-in local user and group databases, and other local configuration files. It stores its configuration in `/etc/nsswitch.conf` … use `more` to take a look.

    Side note: the Mozilla Foundation, makers of Firefox, also make a well-known and widely-used library for authentication and encryption called NSS (network security services) that performs overlapping

functions. A lot of people cross them up, but they're not the same.

It turns out that `nss_ldap` uses the same configuration file as `pam_ldap`. Use `more` to check out `/usr/local/etc/nss_ldap.conf` … indeed, the same file. So let's get rid of `nss_ldap`'s separate version of that file …

```
$ sudo rm /usr/local/etc/nss_ldap.conf
```

… and replace it with a hard link to the `pam_ldap` configuration file.

```
$ sudo ln /usr/local/etc/ldap.conf /usr/local/etc/nss_ldap.conf
```

Note that now they are both **literally** the same file.

```
$ ls -li /usr/local/etc/ldap.conf /usr/local/etc/nss_ldap.conf
1063062 -rw-r--r--  2 root wheel 8730 Feb 26 17:46 /usr/local/etc/ldap.conf
1063062 -rw-r--r--  2 root wheel 8730 Feb 26 17:46 /usr/local/etc/nss_ldap.conf
```

Make your LDAP user's home directory, and make sure they own it.

```
$ sudo mkdir /home/peter
$ sudo chown 1001:1001 /home/peter
```

Now we're going to actually have it start to look at LDAP for users and groups … so here's a great time to SSH into your FreeBSD VM, and start a spare root shell in another terminal tab or window in case things get broken temporarily. Edit the file `/etc/nsswitch.conf` and change both group and password (`passwd`) options from `compat` to `files ldap` … and save it.

Now test, especially before you log out of any root shells, just like you did in lab 1b … `id` and `ls` should be good enough to tell whether it's working or not. FreeBSD is way, way more forgiving if you goof this up. If it failed on OpenBSD, we'd be locked out. If LDAP fails on FreeBSD, we just have to wait for LDAP to time out … at least, that was what I saw.

Test logging into the console of your VM, in its VMware window, as both `failsafe` and your LDAP user. Try logging in via SSH as both users. Test using `id` as in lab 1b to make sure all usernames and group names resolve.

If anything doesn't work, scour through `/var/log` on your FreeBSD VM.

26. Add your LDAP user to the operator group, in `/etc/group` on your FreeBSD VM.

    Add the sudoers group to `/usr/local/etc/sudoers`.

    Create your LDAP user's `.ssh` directory and set up key-based authentication into your FreeBSD VM.

    Configure your environment in your LDAP user account, and stop using the failsafe account. Make sure

your LDAP user can exercise `sudo` rights if needed.

part four: configuring sendmail and dovecot

Typically nowadays, most – including myself – prefer a simpler-to-configure SMTP server, notably `postfix`. However, FreeBSD is pure cane Berkeley Unix and so is `sendmail`, the great-granddaddy of all mail servers. Both `sendmail` and its predecessor, `delivermail`, which used FTP to deliver e-mails, were originally authored by Eric Allman during his time as a student at Berkeley. Eric Allman is married to Kirk McKusick, who he met at Berkeley, and was mentioned in lab 1 for his work on Berkeley's fast file system. Kirk owns the copyright on the BSD daemon, and still works on Berkeley Unix via the FreeBSD project to this day. Eric still works on email, on `sendmail`-derived products to this day, at email security company Proofpoint. The two have been together for forty years, and are kind of the first couple of Berkeley Unix.

So BSD and `sendmail` grew up together, and (sadly) they remained together. OpenBSD didn't replace it with OpenSMTPd until 2013. Why do I say "sadly," you ask? Personal opinion, of course, but for a peek into why I think that, the stock `sendmail` configuration file is almost 60kb in FreeBSD, and it's not really human-readable beyond the first 25% of the file. Fortunately, we build it mostly from pre-configured macro files designed specifically for the purpose of building `sendmail` configuration files.

Within your FreeBSD box, list the contents of `/etc/mail` … this is where all of `sendmail`'s configuration lives.

```
total 1
-rw-r--r--  1 root wheel  6756 Nov 29 02:42 Makefile
-rw-r--r--  1 root wheel  2822 Nov 29 02:42 README
-rw-r--r--  1 root wheel   546 Nov 29 02:42 access.sample
-rw-r--r--  1 root wheel  1612 Nov 29 02:42 aliases
-rw-r--r--  1 root wheel 60145 Nov 29 02:42 freebsd.cf
-rw-r--r--  1 root wheel  4392 Nov 29 02:42 freebsd.mc
-r--r--r--  1 root wheel 41887 Nov 29 02:42 freebsd.submit.cf
-r--r--r--  1 root wheel   912 Nov 29 02:42 freebsd.submit.mc
-r--r--r--  1 root wheel  5988 Nov 29 02:42 helpfile
-rw-r--r--  1 root wheel   247 Nov 29 02:23 mailer.conf
-rw-r--r--  1 root wheel   159 Nov 29 02:42 mailertable.sample
-rw-r--r--  1 root wheel 60145 Nov 29 02:42 sendmail.cf
-r--r--r--  1 root wheel 41887 Nov 29 02:42 submit.cf
-rw-r--r--  1 root wheel   486 Nov 29 02:42 virtusertable.sample
```

`freebsd.cf` is the configuration for a public-facing mail server able to receive mail. This configuration is disabled by default in FreeBSD. `freebsd.submit.cf` is the configuration for the on-by-default internal "submission" mail server that allows you to send mail outbound, provided that you link it up properly to an upstream mail server that will accept your mail. Since over 95% of the e-mail on the internet today is spam, this has become a very particular process, and one of the first checks is whether or not an e-mail contains a legitimate source DNS domain name. Our little .internal network does not have one, so we're not going to be able to get mail outbound … but we're going to learn what that looks like anyways.

The `.mc` files in `/etc/mail` are macro files used to generate the corresponding `.cf` configuration files, using the macro processor `m4`, and the process is still sufficiently complicated that we have a `Makefile` to handle invoking the macro processor and spitting out the file.

27. As we've seen, the `netstat` command can be used for a variety of purposes; we're going to go into it in a little more depth this time.

    The `netstat` command will translate IP addresses to host names using the DNS service and the file `/etc/hosts`, and translate TCP and UDP port numbers using the file `/etc/services`. We've already seen and edited `/etc/hosts` to implement static name service for our VMs on our host system, so now let's check out `/etc/services`.

    TCP and UDP make use of 16-bit port numbers (0 to 65535), but many of these port numbers at the low end are commonly associated with specific services, some of which we're already familiar with: HTTP (web pages), HTTPS (encrypted web pages), DNS ("domain"), SSH, and now SMTP.

    ```
    $ egrep -w "^domain|^http|^https|^ssh|^smtp" /etc/services | more
    ```

    This command should spit out about 20 lines of output from `/etc/services`, matching the four search tokens within the quotation marks (in this context, the vertical bar character is a logical or operator). This file is where a lot of subsystems will look for the *name* of a service associated with a particular port. If the port number doesn't have a name, you'll just see the port number. Of course, this might be what you want at any given point in time.

    ```
    $ netstat -a | grep LISTEN
    ```

    In the above command, the `a` switch asks it for the state of all active network sockets, including those waiting for connections. We're piping this through a `grep` for `LISTEN`, because this is how `netstat` displays the services that are listening for incoming connections.

    ```
    tcp4       0      0 *.ssh                  *.*                    LISTEN
    tcp6       0      0 *.ssh                  *.*                    LISTEN
    ```

    This output tells us that we're running an SSH server on IPv4 and IPv6, on **all** IPs of this system (the asterisk before `.ssh` where the an IP address would regularly go), and that's it … so just the SSH service is currently available to the network. This is sane, and good from a security perspective. Except I was expecting to see this:

    ```
    tcp4       0      0 localhost.smtp         *.*                    LISTEN
    ```

    Had we seen this line, it would tell you that there is a socket/port waiting for connections on the IP address associated with the name `localhost`, on the SMTP service. SMTP is "Simple Mail Transfer Protocol," the language used by mail servers to exchange e-mail messages for almost 40 years now, and `localhost`, is the loopback network adapter on every system, that allows a system to offer network-based services to itself but not to other systems. This would have been the "submission" mail server, configured by `submit.cf` in the directory listing of `/etc/mail` we just saw a minute ago. Since the submission mail server is on loopback, it's not connected to the mail flows on the rest of the internet at large, and it can also only possibly accept mail from other users of the same system (you have to be **on the same system** in some fashion to talk to its loopback adapter). In fact, you should be able to see that some mail has already been delivered.

```
$ ls -l /var/mail
total 1
-rw-------  1 cyrus      cyrus          0 Feb 27 13:33 cyrus
-rw-------  1 dovecot    dovecot        0 Feb 27 13:33 dovecot
-rw-------  1 dovenull   dovenull       0 Feb 27 13:33 dovenull
-rw-------  1 failsafe   failsafe       0 Feb 26 13:50 failsafe
-rw-------  1 git_daemon git_daemon     0 Feb 27 02:03 git_daemon
-rw--w----  1 root       mail        3894 Feb 27 03:47 root
```

`/var/mail` is where incoming email spool files reside for each user, and other programs can be used to read and reply to them in place, or to download the messages to a mail client. Notably here, you can see that on my system, the root user has about 20k of mail waiting. If you also have mail there – that is, if the file size is not zero - you can look through this e-mail easily with the following command …

```
$ sudo more /var/mail/root
```

… and you should see a message warning the root user about your unauthorized use of `sudo` a few steps ago, unless you did something differently. Also, as I told you in lecture, periodic `cron` jobs also send their output via e-mail to the root user, and that means they'll end up in this file.

Like I said, though, I didn't see that line of output … so how was that mail delivered? This is the part of the lab where Peter started to rejoice a bit, after a little bit of confusion. `sendmail` might have finally been vanquished! First, I looked at `/etc/defaults/rc.conf`, which contains the default configuration for a lot of startup services, as we discussed …

```
$ grep sendmail /etc/defaults/rc.conf | grep submit
sendmail_submit_enable="YES"  # Start a localhost-only MTA for mail submission
sendmail_submit_flags="-L sm-mta -bd -q30m -ODaemonPortOptions=Addr=localhost"
```

That made it look like it should be enabled. `ps auxww | grep sendmail` seemed to agree it wasn't running. Then I found this in the FreeBSD 14.0 release notes (link presented previously) …

"The default mail transport agent (MTA) is now the Dragonfly Mail Agent (`dma`(8)) rather than `sendmail`(8). Configuration of the MTA is done via `mailer.conf`(5). `sendmail`(8) and its configuration remain available."

… so it turns out FreeBSD has indeed started to move away from `sendmail` at last as well. `dma`, the Dragonfly Mail Agent, is named for DragonFlyBSD, a fork of FreeBSD where it was created. According to its `man` page, it does its job without opening up network listeners. It all makes sense now.

Take a look at `/etc/mail/mailer.conf` … let's move it out of the way, and replace it with a ready-to-go copy of the file set up for `sendmail`.

```
$ sudo mv /etc/mail/mailer.conf /etc/mail/mailer.conf.dma
$ sudo cp -p /usr/share/examples/sendmail/mailer.conf /etc/mail/mailer.conf
```

I'm sure `dma` is much better than `sendmail` … there's so much room for improvement on the bloat and configuration complexity that comes with `sendmail` – look at `/etc/dma/dma.conf`, 1/30th the size of `sendmail.cf` – but I don't feel like rebuilding the wheel here and now. After you've made this change,

go ahead and `sudo reboot` your FreeBSD VM.

28. Let's save a copy of the distribution's default `.cf` and `.mc` files for a public-facing mail server.

```
$ sudo cp -p /etc/mail/freebsd.cf /etc/mail/freebsd.cf.orig
$ sudo cp -p /etc/mail/freebsd.mc /etc/mail/freebsd.mc.orig
```

The `-p` switch with the `cp` ("copy") command tells it to preserve timestamps, permissions, and ownership of the things it copies … this information is often pertinent, especially when making archival copies of things, in case you make a change that breaks stuff. If you make a mistake and need to come back, of course just reverse the order of the filename arguments in the `cp` command(s).

29. Let's edit the `sendmail` macro file to aim it at `dovecot`, instead of those mail spool files in `/var/mail`. There are client-side mail servers like popper for POP ("post office protocol"), but this is primitive, and we want to be able to set up an e-mail server that lets our user create and manage folders, and synchronize multiple mail clients … so we need IMAP.

```
$ sudo vi /etc/mail/freebsd.mc
```

In this file, you'll notice `dnl` all over the place … this denotes a comment in the `m4` macro processor.

Now find the line that says …

```
MAILER(local)
```

… this is the macro that defines delivery to local mail spool files in `/var/mail`. Just prior to that line, insert the following three lines …

```
FEATURE(`local_procmail', `/usr/local/libexec/dovecot/deliver',`deliver -d $u')
MODIFY_MAILER_FLAGS(`LOCAL', `-f')
MAILER(procmail)
```

… note the ticks are not all the same here, and save the file. We just added delivery to recipient mailboxes via `dovecot` to the macro file that builds the mail server configuration.

30. Now we're going to build a new `sendmail` configuration file from the macro file.

```
$ cd /etc/mail && sudo make freebsd.cf
```

The output from `make` should show you that it's invoking the macro processor `m4`, referring to a pool of `sendmail` configuration information in `/usr/share/sendmail/cf`, processing `freebsd.mc`, and sending the output to `freebsd.cf`. Let's look at the difference between the default `freebsd.cf` (backed up to the `.orig` file above).

```
$ diff /etc/mail/freebsd.cf.orig /etc/mail/freebsd.cf
```

The output of the `diff` ("difference") command uses the greater-than and less-than characters to point "left" and "right" for the two filenames it was handed to compare on the command line above.

You should see some comments at the top of the file (the numbers are line numbers) showing that you built the file locally, as root, in `/etc/mail`, and some additional configuration near the end of the file (it started on line number 1816 for me) to add `procmail` mailer information and `dovecot` information in the place of `/usr/libexec/mail.local`, the program that delivers into `/var/mail`.

31. Install your new `sendmail` configuration file. (Note: you should still be in `/etc/mail`.)

```
$ sudo make install-cf
```

32. We need to tell `sendmail` the domain names it must accept and locally deliver e-mail for, on this FreeBSD VM. We're going to create the file `/etc/mail/local-host-names` …

```
$ sudo vi /etc/mail/local-host-names
```

… and insert the following into it:

```
cs470.internal
openbsd.cs470.internal
freebsd.cs470.internal
```

33. Time to configure `dovecot`.

```
$ cd /usr/local/etc/dovecot && ls -l
```

Note that there's a `README` file here. When you find one, it's there for a purpose.

```
$ more README
```

So … there is a **bunch** of stuff in the directory `example-config`, because we can do a preposterous amount of things with `dovecot` – and I recommend you use `cd` and `ls` to poke around a bit – but we're just going to take a subset of it up a directory level to configure `dovecot`.

```
$ sudo cp -p example-config/dovecot.conf .

$ more dovecot.conf
```

Note that pretty much everything in here is commented out. Lines that are commented out typically contain the compiled-in defaults for each option, and the file ends with statements including everything in a `local.conf` file and `conf.d/*.conf`. This is fairly common nowadays, to break up the configuration files of services into files that contain logically separate subsets of the configuration.

First, let's make a `conf.d` directory here, in the right place …

```
$ sudo mkdir conf.d
```

… and then let's copy out the pieces of the example configuration that we need, leaving the example configuration untouched, in case we need to refer to it again, or to grab a clean copy. Note: the following command will work just fine if you're using `bash`, `csh`, or `tcsh`, but may barf if you're using another shell.

```
$ sudo cp -p example-config/conf.d/{10-auth.conf,auth-system.conf.ext,10-
mail.conf,20-imap.conf} ./conf.d
```

If you're using another shell, you'll almost certainly get a "file not found" error, and will want to separately copy each of the four config files into `conf.d`.

Now let's tweak the files to set up `dovecot` as we want it for our mail server here on our lab networks.

First, edit `dovecot.conf`, and **add** a `protocols` line (leave the sample `protocols` line with all the options intact) just below the sample `protocols` line, saying that we **just** want the `imap` protocol to be served.

In `conf.d/10-auth.conf`, add `login` to the end of the line for `auth_mechanisms`. This tells it that we allow operating system users to log in to the mail service.

Finally, edit `conf.d/10-mail.conf`. Uncomment the line for the setting `mail_location`, and set it like so:

```
mail_location = maildir:/var/spool/imap/%u
```

This tells `dovecot` to create a directory for each user under `/var/spool/imap`, with the username as the name of the folder. Since we've referred to this directory, let's create it, associate it with the `dovecot` subsystem pseudouser, the group `mail`, and allow everybody to write there, but nobody to delete anything that doesn't belong to them. (Remember the sticky bit from lab zero?)

```
$ sudo mkdir -m 1777 /var/spool/imap
$ sudo chown dovecot:mail /var/spool/imap
```

34. Time to make a certificate for `dovecot`. First, let's copy the OpenSSL configuration file over to this system from our OpenBSD VM.

```
$ scp -p openbsd:/etc/ssl/openssl.cnf /tmp/openssl.cnf
```

Move FreeBSD's default OpenSSL configuration file out of the way.

```
$ sudo mv /etc/ssl/openssl.cnf /etc/ssl/openssl.cnf.orig
```

Now, move the file from OpenBSD into its place. Same OpenSSL, same config file, all our defaults in place.

```
$ sudo mv /tmp/openssl.cnf /etc/ssl/openssl.cnf
$ sudo chown root:wheel /etc/ssl/openssl.cnf
```

We're not done, though … `/etc/ssl/openssl.cnf` says to request certificates with the subjectAltName of our OpenBSD VM. Edit the file, and change the hostname at the very end to `freebsd.cs470.internal`.

35. Time to generate a key … first, make a root-only private directory …

```
$ sudo mkdir -m 700 /etc/ssl/private
```

… generate a key …

```
$ sudo openssl genrsa -out /etc/ssl/private/freebsd.key 3072
```

… and then generate a certificate request.

```
$ sudo openssl req -new -key /etc/ssl/private/freebsd.key -out /etc/ssl/freebsd.csr
```

You should recognize the next steps. For the common name, use `freebsd.cs470.internal`. Once the CSR file is cooked, copy it over to your OpenBSD VM for the CA to sign it.

```
$ scp -p /etc/ssl/freebsd.csr openbsd:/home/pki/newreq.pem
```

Consider yourself warned: at some point soon here, I'm going to stop spoon-feeding you commands for moving files around. You should be able to infer the CSR filename from the prior command, how to use your PKI commands, and how to do various things from prior labs.

Now, log into your OpenBSD VM as your LDAP user and sign the certificate.

```
$ cd /home/pki && CA.pl -sign
```

Take a moment to validate the CSR has all the information you need entered into it correctly, especially the hostname in both places (the common name and the subjectAltName), and sign the certificate. Copy it back from your OpenBSD VM to your FreeBSD VM.

```
$ scp -p newcert.pem freebsd:/tmp/freebsd.pem
```

Again, clean up the new certificate and the CSR here on OpenBSD VM, as in lab 1b.

Back on your FreeBSD VM, let's get the certificate into the right place and into the `dovecot` configuration.

```
$ sudo mv /tmp/freebsd.pem /etc/ssl/freebsd.pem
```

Set its permissions properly.

```
$ sudo chown root:wheel /etc/ssl/freebsd.pem
```

Edit `/usr/local/etc/dovecot/dovecot.conf` again, and add the following lines near the front of the file, right after the `protocols` line you set up earlier.

```
ssl = required
ssl_cert = </etc/ssl/freebsd.pem
ssl_key = </etc/ssl/private/freebsd.key
ssl_ca = </etc/ssl/cacert.pem
```

We could also use this certificate to encrypt e-mail to `sendmail` as it's in transit between SMTP clients on the other VMs and our target SMTP recipient for everything here on FreeBSD. The certificate we just set up for the IMAP service protects passwords, but we're not e-mailing any state secrets in `cron` jobs between our VMs, nor do we have to authenticate for that server-to-server mail traffic, so we'll take a pass.

36. Let's set up authentication for `dovecot`. `dovecot` would always complain in prior iterations of the class about not having a `pam.d` file, but this time our client-side connection issues didn't clear up until we did this:

```
$ sudo cp -p /etc/pam.d/sshd /etc/pam.d/dovecot
```

Edit the file and make sure all configuration line references to `ssh` are commented out – we don't want to use it to log into the IMAP server – and change the header comment too.

37. Now, enable both `sendmail` and `dovecot`. Edit `/etc/rc.conf` and add the following three lines:

```
dovecot_enable="YES"
sendmail_enable="YES"
sendmail_flags="-bd -q30m"
```

The `sendmail` flags tell it to become a daemon (a server, `-bd`), and to run its queue every 30 minutes (`-q30m`) to try to redeliver mail that somehow gets stuck there.

Now, let's start up both services.

```
$ sudo /etc/rc.d/sendmail start
$ sudo /usr/local/etc/rc.d/dovecot start
```

Note that `sendmail` is started from scripts in `/etc/rc.d`, because it was bundled with the operating system. `dovecot` is started under `/usr/local`, because it was installed from the ports tree. We are restarting `sendmail`, because some of its components were already running (remember the submission mail server?), but `dovecot` has never run before.

Run `netstat -a | grep LISTEN` again. Look a little different? How about if you add in the `n` switch?

38. Test your mail server.

```
$ echo "test" | mail -s test peter
```

Naturally, substitute in your username for mine. Now check the mail subsystem log.

```
$ sudo tail /var/log/maillog
```

`tail` shows the end of a file, in this case, the log for the mail subsystem. You should see a line showing the mail being received by the submission `sendmail` instance, another by the other `sendmail` we set

up, accepting it for delivery, and finally, `dovecot` delivering it into your user's inbox. Listing the contents of `/var/spool/imap` should also show that some stuff has been created.

39. Let's configure your OpenBSD VM's SMTP server, so that it's able to send out mail. **On your OpenBSD system**, use `sudo` and `vi` to create the file `/etc/mail/mailname` … this file did not exist on my OpenBSD VM prior to this step … and put `openbsd.cs470.internal` into it. That's it, nothing else. Just one line, just your OpenBSD VM's fully-qualified DNS domain name, then save it out.

    These last two things, adding an MX record for the OpenBSD VM and editing `smtpd.conf`, I had to do in 2020, didn't have to do in 2021, and did have to do in 2022. Neither proofreader required these last two steps for a working setup this year. If you don't mind, try only setting `mailname`, skip below to the `rcctl` command, and see if it works overnight.

    Also on your OpenBSD system, edit the zone file for cs470.internal, increment the serial number, and add a MX record for your OpenBSD VM, pointing to your FreeBSD VM. After you make this change, restart `named`.

    Still on your OpenBSD VM, edit `/etc/mail/smtpd.conf` … and let's comment out the first *uncommented* line beginning with "match from local for local"… that is, insert a pound sign (#) at the beginning of the line. Here, we tell `smtpd` that there is no local mail delivery, period.

    Save, and restart `smtpd` to make sure it notices the changes we just made …

    ```
    $ sudo rcctl restart smtpd
    ```

40. Let's tell this FreeBSD system who its sysadmins are; for a variety of reasons, you don't want root checking mail.

    ```
    $ sudo vi /etc/mail/aliases
    ```

    The aliases file is a list of mail re-direction rules. You'll notice that e-mail addressed to pretty much every subsystem of our FreeBSD VM is routed to root. However, all that mail going to root – it's not all that much – but we don't want it going to the root account's mail spool. We want it to go to the system administrators of our system, each of us, and NOT to have to log in as root via a mail server to grab it.

    There's a comment about just this, forwarding root's e-mail, and under that a line that looks like this:

    ```
    # root: me@my.domain
    ```

    Uncomment it (remove the leading # and space), and replace me@mydomain with username for your LDAP account on your FreeBSD system @cs470.internal (the domain for our private networks). For me, it looked like this:

    ```
    root: peter@cs470.internal
    ```

You now know what the at-sign is for in an e-mail address, and how the whole system behind that works.

When we edit the aliases database, we need to tell the mail subsystem to parse it and re-generate the database file that sendmail actually references during regular operation (note that there's an `aliases.db` file in `/etc/mail`).

```
$ sudo newaliases
```

Now, do the same alias configuration (all of the prior step) **on your OpenBSD VM**. The `aliases` file is in the same place, you want to substitute in the same data, and you have to run the `newaliases` command afterwards.

41. Now test a mail to root on each VM, and verify it works.

```
$ echo "test" | mail -s test root
```

Also test e-mail to your LDAP user on each VM.

```
$ echo "test" | mail -s test peter
```

Also test e-mail to your local-network e-mail address on each VM.

```
$ echo "test" | mail -s test peter@cs470.internal
```

You can `tail /var/log/maillog` – you may need to use `sudo` – on both your OpenBSD and FreeBSD VMs to see whether or not your test worked through through mail servers. You should see your FreeBSD VM listed as the "relay" and "message accepted for delivery" on OpenBSD. On the FreeBSD side, you should see `sendmail` accept the mail, then `dovecot` save it into an inbox.

Since the e-mail address we're supplying in our test case here is @cs470.internal, the SMTP service on OpenBSD (OpenBSD has its own `smtpd`, and doesn't use `sendmail` or `postfix`) goes out and consults DNS to find out where to deliver e-mail for the domain "cs470.internal." It is configured to be its own name server, and if you look back at what we set up in DNS, we configured an MX record (MX is short for "mail exchanger") for the domain cs470.internal, pointing at our FreeBSD system.

On both VMs, run the following command and see what it says …

```
$ host -t mx cs470.internal
```

… and run it for your OpenBSD VM by FQDN as well.

```
$ host -t mx openbsd.cs470.internal
```

It should report that the preferred mail exchangers for both cs470.internal and openbsd.cs470.internal are your FreeBSD VM, unless you're trying to get it working without a second MX record. E-mail should

now be flowing properly between our first two VMs. Check mail logs at both sides if it's not working, and start from error messages there.

part five: configuring regular maintenance, diagnostic mails, and patching

42. Let's recoup some resources. Run `top` and look at its output and see how much RAM your VM is now not using. My AMD64 FreeBSD VM showed 798MB free; my ARM64 VM showed 693MB of RAM as "inactive." Based on this, I shut down and moved both architectures' FreeBSD VMs down to 256MB RAM.

43. Back on our FreeBSD VM, let's set up some `cron` jobs.

    ```
    $ sudo crontab -e
    ```

    Add in the following lines:

    ```
    0 3 * * * /usr/local/sbin/pkg update
    0 4 * * * cd /usr/ports && /usr/local/bin/git pull --ff-only
    0 5 * * * /usr/sbin/freebsd-update cron
    0 6 * * * /usr/local/sbin/pkg version -l "<"
    ```

    These four lines will automatically update the package database at 3 AM, pull down updates to the ports tree at 4 AM, check for updates to FreeBSD (the base operating system) at 5 AM, and finally, give you a report of packages (that switch after `pkg version` is the first letter of the word "list") at 6 AM. All four jobs will run every day.

    If your ports tree has issues updating with `git`, I recommend this page: https://people.freebsd.org/~dch/git.html

    The `pkg version` command will match packages for which an updated pre-compiled binary package is available online **or** if the ports tree knows how to build a newer version. You can install updates via `pkg` with the `upgrade` verb.

    Often time, the package repositories lag behind the ports tree, and this may leave you needing to update software via the ports tree. If that's the case, often times what you need to do is to go to the port directory and run `sudo make uninstall && sudo make reinstall`. The ports called `portmaster` and `portupgrade` can also help with this task.

    !! IMPORTANT NOTE: if `ca_root_nss` gets updated at any point and LDAP or your certificates stop working on this VM, you may need to take a look at what you did there.

    The ports tree can end up become quite a disk hog after you've use it for a while … don't forget your friend `du` from lab zero for finding which file trees consume the most storage. You can issue `sudo make clean` at any level of the ports tree to remove old build trees below that directory and reclaim space.

‼ REMINDER: I want you to start running your VMs all night, as much as possible, for the duration of the class, so that these late-night `cron` jobs turn up some output, and start e-mailing it to you. You will not be graded on uptime. If you'd rather save on your electricity bill, feel free to change the `cron` jobs to run by day.

44. Please take a look at `man freebsd-update`.

    Then, go ahead and patch your FreeBSD VM's base operating system immediately. This involves first inspecting our system to determine which (if any) updates to fetch from the FreeBSD team's systems, then installing them. Fortunately, `freebsd-update` allows us to supply multiple verbs on a single command line.

    ```
    $ sudo freebsd-update fetch install
    ```

    As I sit here and type this, there are already patches available for FreeBSD 14.2.

    ```
    $ sudo freebsd-update fetch install
    Looking up update.FreeBSD.org mirrors... none found.
    Fetching public key from update.FreeBSD.org... done.
    Fetching metadata signature for 14.2-RELEASE from update.FreeBSD.org... done.
    Fetching metadata index... done.
    Fetching 2 metadata files... done.
    Inspecting system... done.
    Preparing to download files... done.
    Fetching 50 patches.....10....20....30....40....50 done.
    Applying patches... done.
    The following files will be updated as part of updating to
    14.2-RELEASE-p2:
    /bin/freebsd-version
    /boot/kernel/cd9660.ko
    /boot/kernel/ext2fs.ko
    /boot/kernel/kernel
    /boot/kernel/tarfs.ko
    /usr/bin/slogin
    /usr/bin/ssh
    /usr/bin/ssh-add
    /usr/bin/ssh-agent
    /usr/bin/ssh-keygen
    /usr/include/isofs/cd9660/iso.h
    …
    ```

    When patching in the future, you'll likely just have to supply the `install` verb, as the overnight job we set up with `crontab` will automatically download any available updates for you, and send you an e-mail to let you know of their availability.

    How will you get that e-mail? Glad you asked.

part six: configure an e-mail client

Mac users, use Mail.app. Windows and Linux users, I recommend getting Thunderbird … it's free, way better than the Windows mail client, and far easier to configure than Outlook, if you're on Windows. Linux users, you probably have Thunderbird installed along with your operating system. If you don't, go get it.

No matter which operating system you're on, set up a mail client to get e-mail from the IMAP server on freebsd.cs470.internal … because we set this up in `/etc/hosts` on your host operating system, your mail program should be able to aim there without changing your host operating system's DNS configuration to aim at your OpenBSD VM.

If you get certificate warnings, check them out! You should <u>NOT</u> get any certificate errors if you did everything correctly.

If you get your earlier test e-mails in your mail client, mission accomplished.

Other problems here setting up a mail client? I want you to web search them. Whatever problem you have setting up that e-mail program, I guarantee you're not the first person to have it. I've spoon-fed you enough for today.

`</lab2>`