CS 470: Unix/Linux Sysadmin
Spring 2025 Lab 1b
LDAP directory and encryption certificates with OpenSSL

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

- lab 1: working DNS setup with BIND, resolving cs470.internal and internet hostnames

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

- getting your certificate authority going and issuing certificates
- getting LDAP to work for authentication and user/group name lookups

Your smörgåsbord at the knowledge buffet continues. Loosen your belt a bit to make room, but also make sure you get some exercise in, lest all of this know-how go straight to your waistline.

In this addition to the OpenBSD lab, we stand up more services on our OpenBSD VMs: first, a certificate authority ("CA") to provide a trust and encryption framework for mutual client-server authentication and data security across our lab networks, and LDAP to provide a common directory of user accounts and groups, and centralized and federated authentication services across our VM network.

Your OpenBSD VM will always need to be up for your lab to work properly. Not only will it be required to access your labs systems and hosts on the internet by name with its DNS service, but it will also be required as an identity provider for your other VMs. The primary user account that we will use for all labs moving forward will be authenticated using the LDAP service we're going to set up here in lab 1b. We'll put a "failsafe" user on each system, **but it is intended only for use during setup and in case of emergency, like the LDAP service breaking. After you get LDAP working on each system, you are to stop using its failsafe account.**

We're only going to set up one user and a couple groups in LDAP, to show you how it's done. Please feel free to experiment and create other accounts. Just make sure implement the account and groups I ask for, as directed. This is what I'll be grading you on later, of course.

When designing this lab, I briefly considered what once might have been the ultimate heresy: standing up a Windows server in this Unix class to provide LDAP via an Active Directory domain, because this is how it's done in most modern IT departments. Using Active Directory on Windows allows for login credentials, access controls, and authorization databases to be shared by Windows, plus all other platforms that know how to subscribe to it. However, I decided doing so would represent a bridge too far, given the name of the class, the graphical user interface on Windows, and the hardware requirements announced at the beginning of the class. A Windows server would have pushed the RAM requirement for our lab host computers above 8 GB in the days that we were able to support that little RAM, and would have forced many to go shopping.

I do, however, still recommend you log into Azure for Education to grab Windows Server and all the free loot you can get while still a student. Experiment with Windows domains and joining Unix and Linux systems to them later, on your own time, or see me in my advanced security class where we'll be standing up an Active Directory and joining Unix systems to the domain.

part one: certificate service (PKI root CA)

Setting up a certificate service, often also called a "certificate authority" or a "CA," is very much like setting up a government agency similar to the Department of Motor Vehicles, and issuing driving licenses and other forms of generally-accepted identification. When selling alcohol or cigarettes, voting, or letting somebody on an airplane, it's not a good idea to just take somebody's word for their identity. Certificates also address the problem of trusting vendors across the internet, when there's no storefront door to walk through to establish trust before providing critical information. Just because that website looks like it belongs to your bank, it's not necessarily enough assurance that we should feel safe typing in our online banking username and password.

In both cases, we need a trusted third party to provide some assurance that a client or vendor is an entity we want to transact with, and that they are the entity they claim to be. As much as possible, we'd like to have some semblance of assurance that's our bank website, or really one of our company's computers, before we provide a username and password. Certificates allow mutual identity verification to happen. Certificates also facilitate encryption to protect the confidentiality of sensitive information, like login credentials, while they're in transit over the network.

Certificates make use of several properties of asymmetric encryption algorithms, also called "public key" encryption. This is because, in asymmetric algorithms, two keys are generated for each subscribing entity (a person, service, or system). What one key does, the other un-does, and it should be mathematically infeasible to figure out one key from the other, in either direction. One key is designated public at the time of generation, and is treated as public information while the other is the private key and is never disclosed. The disclosure of a private key would force a user to generate new keys, so we often use devices like smart cards, tokens, USB fobs, or special-purpose chips to generate and contain the keys. Generated this way, a private key is more difficult to compromise, even when held by unsophisticated end users.

By having one key public, anyone subscribing to a public key infrastructure can encrypt messages to anyone else such that only the intended recipient(s) can read them, because only the recipient has the matching private key. The private key can be used to encrypt data as well; because that data can only be decrypted by the associated public key, data encrypted by a private key can be reasonably assured to have come from the person or system to which the public key belongs. This is often called a cryptographic or digital signature.

Thus, we are able to associate a public key with a person, system, service, or entity because a trusted third party (the certificate authority) validates their identity. This provides a sense of assurance to whom the public key was issued by cryptographically signing the key and all associated metadata we'd like to validate. The public key is given an expiration date, because we need to rotate key materials periodically. This is combined together in a structured data format with associated identity information like a DNS hostname for a server or website, a name and e-mail address for a person (servers may be associated with people too), and allowed uses for the key pair. All this is combined together as a single, standardized document and digitally signed by the CA. A certificate is this signed combination of public key, identity, allowed uses, effective dates, and other metadata.

This identity validation function of key infrastructures is publicly provided by a number of companies with sufficient reputation and security procedures, such that their certificates are shipped in caches of what are called "trust anchors" along with a lot of web browsers and operating systems. However far and wide the root certificate authority's signing certificate in any chain of trust is distributed with people and systems to trust it

… is how far certificates signed by it will be accepted. If one of these public CAs signs our entity's root signing certificate, we become subordinate to its chain of trust, but certificates we sign can be validated using their CA certificates already in our browser or operating system "trust anchor" databases.

You've probably seen the effect of not having a trusted certificate here or there, when pulling up an HTTPS management interface on a new device with a self-signed certificate … in this case, your browser will tell you "unknown authority," "issuer unknown," or something like that. Every once in a while you'll run into an encrypted website with a similarly self-signed certificate, and that's okay if you just have to read data from it but don't have to authenticate to it. A more common one is a website that hasn't renewed its certificate before its expiration date; your browser will tell you "certificate expired," and ask if you want to continue. Of course, you don't want your end users experiencing any of these errors or warnings before they do important work, and this makes it important to link to a publicly-known chain of trust, or to provide our own internally.

When we stand up a certificate authority internal to an entity or a business, we often just distribute that certificate within the organization, effectively having the organization itself behave as the identity authority for local systems and users. This provides a local means of validation, often at a lower cost. Having a certificate signed by a public CA typically costs tens of thousands of dollars in fees and hardware, to assure a subordinate organization doesn't create problems for the overall chain of trust. Signing your organization's root CA by a public CA, however, saves a lot of labor pushing around a custom chain of trust to all browsers, services, and systems … and labor is one of the significant costs in wide deployments of key infrastructures.

Each major browser or platform has a cache of trust anchors.

- macOS's built-in certificate store. In `/Applications/Utilities`, open up `Keychain Access.app` and click on "System Roots." "Roots" here refers to the root of a chain of trust.
- On Windows, type `CertMgr` into the start menu to start the Certificate Manager utility, then click "trusted root certification authorities."
- For Linux, we'll dig into this later in the labs, but for Debian-like operating systems like Ubuntu, there are directories of trust anchors under `/usr/share/ca-certificates` and `/usr/local/share/ca-certificates`. The tool `update-ca-certificates` is used to rebuild the database of trust anchors. Both FreeBSD and Rocky Linux configure very much like OpenBSD does in this lab.
- In Firefox, go to preferences (`about:preferences` in the browser's location bar), then "privacy & security," then scroll all the way to the bottom and click "view certificates" and "authorities."
- Edge uses the default certificate store for your operating system; see above for Mac or Windows.
- If you're using another browser, just do a web search for how you find and examine your trust anchors.
- If you're using Chrome or Chromium, you're using **spyware** and should seriously consider another browser, unless you don't mind being spied on, having your behavior aggregated by Google, and sold to advertisers. Don't take my word for it, though.

  https://www.forbes.com/sites/zakdoffman/2021/03/20/stop-using-google-chrome-on-apple-iphone-12-pro-max-ipad-and-macbook-pro/
  https://www.forbes.com/sites/zakdoffman/2021/08/28/stop-using-google-chrome-on-windows-10-android-and-apple-iphones-ipads-and-macs/

Microsoft's Edge browser is based on Chromium, it does *almost* the same user tracking … only the data goes back to Microsoft. I don't recommend this browser either.

As stated above, each trust anchor is the root of a chain of trust. Each "root" certificate authority also typically creates a number of subordinate certificate authorities. The trust relationship is created by means of digital signature … superior certificate authorities "sign" the keys of subordinates, and trust in the superior is passed on to its subordinates, and allows its subscribers to trust each other's key material by means of the certificates signed by the subordinates, because the chain of trust can be validated.

With big certificate authorities, typically only "intermediate" authorities stay online to do the work of automated issuance as a service; this is to divide up the chain of trust. If a certificate authority's key gets compromised, its certificate needs to be revoked … and so do all the certificates signed by that certificate authority ("CA"), and all the certificates signed by certificates signed by that certificate authority and so on, because they could **all** be forged. Root certificate authorities are typically kept offline – **not** on the network – and only powered up in order to sign the creation of intermediate certificate authorities, then pulled back offline so that it is far harder to compromise them and of course, their private key material. If a root CA's private key is compromised, the trust of the whole infrastructure is instantly killed and all certificates have to be re-issued … something nobody wants to go through.

As mentioned, it can cost lots of money and time to roll out a PKI, especially if it's larger in size or in scope, and it's for this reason that many entities have been slow to roll out key infrastructures of their own. Most companies get a certificate for their public web server from a public CA service, and call it a day for encryption.

As usual, Wikipedia has some good reading on the subject.

- https://en.wikipedia.org/wiki/Certificate_authority
- https://en.wikipedia.org/wiki/Root_certificate

Note that we often use the terms "SSL" and "TLS" interchangeably to describe the family of protocols used for establishing encryption, trust, and authentication over arbitrary network connections using certificates. The first three versions of the protocols were called SSLv1, SSLv2, and SSLv3. Ground-shaking security defects were found in each successive version of the protocol, and when the time came to replace SSLv3, a complete re-write was in order, and the successor was named TLSv1.0. Ground-shaking flaws have also been discovered in TLSv1.0 and TLSv1.1, neither of which should be used in production anymore. Only TLSv1.2 and TLSv1.3 should be used for communications we'd like to keep safe.

As briefly touched on in lab 1, OpenBSD forked the source for OpenSSL in order to create a leaner variant, LibreSSL. Unfortunately, part of what they left on the chopping block from OpenSSL was a lean command-line certificate authority, and that's exactly what we need right now. We're going to grab it back from the OpenSSL project's source tree directly.

1. SSH into your OpenBSD VM. Make a directory for your new public key infrastructure, and for binaries, data, man pages, and source files underneath it …

   ```
   $ sudo mkdir -p /home/pki/{bin,data,man/man1,man/man5,src}
   ```

   … and make it group-writable recursively.

```
$ sudo chmod -R g+w /home/pki
```

Note that by default, directories created by root, and created by users borrowing rootly powers using `sudo`, are created in root's default group, which is wheel on OpenBSD. We are making the system administrators of the OpenBSD system able to change the PKI folder. This might not be appropriate in a business, but here on our little lab network, it makes sense.

Now make your PKI directory inaccessible to others, besides the root user and the wheel group …

```
$ sudo chmod o-rwx /home/pki
```

… we're not using the `-R` flag here for a recursive `chmod`, because if all permissions are blocked on `/home/pki` then nobody outside the owner (root) or group (wheel) can get to anything beneath it anyways.

2. Open up a web browser and go to openssl-library.org, then click on download. Right-click on and copy the URL for the **source code** of the latest version of the OpenSSL 3 release train. As of the time of the writing of this lab, this was OpenSSL 3.4.0.

   `cd` into `/home/pki/src` and use OpenBSD's FTP client to download the source ball for OpenSSL, with the link you copied above. Unpack the OpenSSL sources with `tar` …

   ```
   $ tar zxvf openssl-3.4.*.tar.gz
   ```

   … `cd` into the OpenSSL source tree, and use `find` to discover files associated with `CA.pl`, OpenSSL's easier-to-use command line certificate authority.

   ```
   $ find . -iname "CA.pl*"
   ```

   You should see two files, `CA.pl.in` and `CA.pl.pod` and should notice one under `apps` and the other under `doc/man1` because it's documentation.

   Copy `CA.pl.in` to `/home/pki/bin` and copy `CA.pl.pod` into `/home/pki/man/man1`.

   Also, find `config.pod` and copy it to `/home/pki/man/man5/openssl.cnf.pod`.

3. The manual pages end with `.pod` because `CA.pl` is written in Perl (`.pl`) and POD is Perl's native documentation format. OpenBSD comes with a tool, `pod2man`, to convert `.pod` files to man pages …

   ```
   $ which pod2man
   /usr/bin/pod2man
   ```

   … do the thing to make them into proper `man` pages …

   ```
   $ pod2man /home/pki/man/man1/CA.pl.pod /home/pki/man/man1/CA.pl.1
   $ pod2man /home/pki/man/man5/openssl.cnf.pod /home/pki/man/man5/openssl.cnf.5
   ```

… and add a line to ~/.profile on your OpenBSD VM to add /home/pki/man to the end of your MANPATH. Also, add MANPATH to the list of exported environment variables already in your default .profile file.

```
MANPATH=$MANPATH:/home/pki/man
```

After that, use source to read in your .profile …

```
$ source ~/.profile
```

… and then you should be able to check out the man page for CA.pl …

```
$ man CA.pl
```

… and for the master OpenSSL configuration file.

```
$ man openssl.cnf
```

4. CA.pl.in is named as such because it needs to be configured for the local computer. See the first line of CA.pl.in with head.

```
$ cd /home/pki/bin && head -1 CA.pl.in
#!{- $config{HASHBANGPERL} -}
```

As you have hopefully already inferred, CA.pl is written in Perl, which is an interpreted language. Since you better have already done some light reading in Kochan, you're aware that interpreted files, including shell scripts, have a first line called the hash-bang, or shebang, that turns a script into an executable by aiming the operating system at its interpreter. Copy CA.pl.in to CA.pl …

```
$ cp /home/pki/bin/CA.pl.in /home/pki/bin/CA.pl
```

… and now we're going to make a few necessary edits to it. In CA.pl, replace the first curly bracket and everything after it on line 1, with the full path of perl in OpenBSD. You should know how to find this by now. The link just above might be instructive.

Also, please find the line in CA.pl that defines the variable CATOP and replace ./demoCA with /home/pki/data.

Finally, find the line that begins to define the command for a new certificate authority (you should see it right after a line advising the operator of CA.pl that it is "Making [a] CA certificate …") and add the missing text below have the new CA key pair use the RSA algorithm and 4096 bits of key …

```
$RET = run("$REQ -new -newkey rsa:4096 -keyout ${CATOP}/private/$CAKEY"
```

… anything less would be uncivilized for a root certificate key pair. Now, save CA.pl back out to storage, and then make it executable.

```
$ chmod ugo+x /home/pki/bin/CA.pl
```

Add `/home/pki/bin` to the end of your `PATH` in `~/.profile`, `source` it again, and make sure it's now in your `$PATH`.

5. Although LibreSSL has forked OpenSSL, a lot of its files, and its primary command line utility still retain the names from the parent project. Move the default OpenBSD SSL config file out of the way …

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

   … and use `find` again to find `openssl.cnf` under the OpenSSL source tree you unpacked. Use `sudo` and `cp` to copy that `openssl.cnf` to `/etc/ssl/openssl.cnf` … then open it up in `vi`.

   As you'll see, `openssl.cnf` is broken up into configuration stanzas, with headers denoted by hard brackets (`[` and `]`), a format popularized under Windows. Comment out the line near the top that sets `openssl_conf` to automatically load providers. Under `CA_default`, change `dir` to `/home/pki/data`. Under `req_distinguished_name`, change the default country name to `US`, the default `stateOrProvinceName` to `Californistan`, and the default `organizationName` to `SDSU CS470`.

   OpenBSD, and security and encryption software like OpenSSL, are often finicky about file ownership, so let's give our new `openssl.cnf` to root:

   ```
   $ sudo chown root /etc/ssl/openssl.cnf
   ```

6. Use `CA.pl` to create a new certificate authority.

   ```
   $ CA.pl -newca
   ```

   `CA.pl` will prompt you immediately for a CA certificate filename, or to hit `enter` to create one, so hit `enter`. You will see it generate an RSA private key …

   ```
   Directory /home/pki/data exists at /home/pki/bin/CA.pl line 158.
   CA certificate filename (or enter to create)

   Making CA certificate ...
   ====
   openssl req  -new -newkey rsa:4096 -keyout /home/pki/data/private/cakey.pem -out
   /home/pki/data/careq.pem
   Generating a 4096 bit RSA private key
   ..........................
   ..................
   writing new private key to '/home/pki/data/private/cakey.pem'
   Enter PEM pass phrase:
   Verifying - Enter PEM pass phrase:
   ```

   It's asking for a passphrase (password) to protect your new CA's private key. Pick a good one. Then it will ask you to fill out information to be used in the certificate. Because you provided defaults in the prior step, you should see them in the next part of the process, for the country name, state, and organization, and be able to leave a blank line on each of those options to select the defaults.

   You can leave the city/locality and organizational unit name blank. For the Common Name, put in `CS470 2025 Spring CA`. You can leave the locality name, organizational unit, email address, and challenge password all blank.

This was what my data entry looked like.

```
Country Name (2 letter code) [US]:
State or Province Name (full name) [Californistan]:
Locality Name (eg, city) []:
Organization Name (eg, company) [SDSU CS470]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:CS470 2025 Spring CA
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

It should then ask you for your key passphrase, generated just above, and if everything went as planned, you should end up with a new CA certificate inside `/home/pki/data/cacert.pem` …

```
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            e8:ce:43:c6:88:92:7f:1d
        Validity
            Not Before: Feb  8 20:51:44 2025 GMT
            Not After : Feb  8 20:51:44 2028 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = Californistan
            organizationName          = SDSU CS470
            commonName                = CS470 2025 Spring CA
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                AC:48:5B:D9:D0:89:76:E5:90:36:C0:C2:3B:FB:D2:AB:2B:FB:45:9E
            X509v3 Authority Key Identifier:
                keyid:AC:48:5B:D9:D0:89:76:E5:90:36:C0:C2:3B:FB:D2:AB:2B:FB:45:9E

            X509v3 Basic Constraints: critical
                CA:TRUE
Certificate is to be certified until Feb  8 20:51:44 2028 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
==> 0
====
CA certificate is in /home/pki/data/cacert.pem
```

7. Add the new CA certificate to the trust anchors database for OpenBSD. First, back up that file, just in case anything goes wrong.

```
$ sudo cp -p /etc/ssl/cert.pem /etc/ssl/cert.pem.orig
```

Then invoke a root shell …

```
$ sudo bash
```

… because we need to add our new CA certificate to the end of that file. If you run a command with `sudo`, `sudo` does not make I/O redirection happen as root … that still happens as you, in your shell.

This gets around that. Note: the double less-than (>>) makes sure that we append to the file, adding our certificate to the operating system's list of trust anchors, rather than truncating it before writing to it.

```
# cat /home/pki/data/cacert.pem >> /etc/ssl/cert.pem
```

When I first wrote this step of in the Summer of 2022, my son Jack was there in the room watching me, and said something to the effect of "Really, dad, it uses a cat to show the contents of a file?" Yes, it does. `cat` is short for "concatenate," and to show him this, I used `man -k` on EDORAS – which I was then logged into – to show him that `cat` printed out files, which started with this output …

```
$ man -k cat | grep -w cat
cat (1)              - concatenate files and print on the standard output
cat (1p)             - concatenate and print files
dog (1)              - better than cat
…
```

… I might have missed it, but Jack loves cats, and he was gobsmacked when he saw the third line, pumping his fists in the air with a "NOOOOOO." It turns out somebody wrote a replacement for `cat`, and of course named it `dog`. First and foremost, one of the great debates of **all time** has been settled. This also serves continues a long-time userland tradition of giving funny derivative names to tools intended to replace or unseat popular incumbents. When somebody went to rewrite `more`, they naturally named the intended replacement `less`. Also using `man -k` on EDORAS, you can see that `less` is the opposite of `more` …

```
$ man -k less | grep -w ^less
less (1)             - opposite of more
less (3pm)           - perl pragma to request less of something
```

… but here on your OpenBSD VM – remember what I told you in lab zero and in lecture about hard links – `less` literally **is** `more`. Yet another one of the great debates, resolved.

```
$ ls -li `which more` `which less`
1322024 -r-xr-xr-x  2 root  bin  150224 Sep 30 07:33 /usr/bin/less
1322024 -r-xr-xr-x  2 root  bin  150224 Sep 30 07:33 /usr/bin/more
```

Not to be outdone, when compiler people grew tired of limitations with Yet Another Compiler Compiler (`yacc`) and wanted a GPL-licensed alternative, they replaced it with `bison`.

```
bison (1)            - GNU Project parser generator (yacc replacement)
```

I once read somewhere, that `bison` was actually a backronym for Bob's Iconographic something something, but can no longer find that. If you feel inclined to look, and find it, please let me know.

8. From your host operating system instance, use `scp` to grab the new CA certificate to your host operating system. The command below will stick it in your Downloads folder, if you're on a Mac or a Linux system.

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

If you're on Windows, you'll want to copy it outside the WSL file namespace to somewhere you can more easily get to from the rest of Windows. The below command is a single line, wrapped, and of course you'll have to substitute your Windows username for mine for the right home directory under `/mnt/c`.

```
$ scp -p failsafe@openbsd:/home/pki/data/cacert.pem /mnt/c/Users/peter/Downloads
```

Mac users, use `Keychain Access.app` to add the certificate in `cacert.pem` to your login keychain. Find the certificate after importing it, and make sure the certificate is trusted for all use cases. After you're done, Keychain Access should show "this certificate is marked as trusted for this account."

Windows users, rename the certificate file `cacert.crt`. You should then be able to right-click the certificate in File Explorer, and have an option to "Install Certificate." Import it for the local machine, and choose to place the certificate into the "Trusted Root" CAs store. Use the Certificate Manager, as mentioned above, to make sure your computer sees it as a trusted root.

Linux/Firefox users, import the certificate as a trusted authority in Firefox. If you're using another browser, figure out what to do here. A web search is your friend, so long as that web search is not served up by Google.

9. Use `sudo` and `vi` again to edit `/etc/ssl/openssl.cnf` … we're going to make a few more changes. What we had in there was appropriate for signing a CA certificate, but we have more work to do to be standards-compliant for the server certificates we'll be issuing through the rest of the labs.

   Many years ago, it was mandated that identifiers like DNS hostnames and e-mail addresses for the subject of a certificate should go into an extension in the certificate, called subjectAltName, so that the common name could be used for any plain-language description, and subjectAltName could be used for the many names a subject often has, whether multiple e-mail addresses, multiple hostnames, or something else entirely.

   Find the line that defines the extensions to add to a certificate request, and uncomment `req_extensions = v3_req`.

   Then find the `v3_req` section header that we just added to the mix by uncommenting that line, and add the following three lines into that section to make sure that certificates we request from here on out on this system are good for mutual client/server authentication, and need a subjectAltName:

   ```
   subjectKeyIdentifier = hash
   extendedKeyUsage = serverAuth,clientAuth
   subjectAltName = @alternate_names
   ```

   Find the line with `copy_extensions = copy` and uncomment it. Note the warning here; this isn't something we want to uncomment lightly. If we weren't checking certificate requests thoroughly, and others were issuing them, this might not be something you'd want to do. However, here on our lab network, only we are issuing certificate requests, so it's okay under these circumstances.

   Finally, add a new section at the very end of the file …

```
[ alternate_names ]
DNS.1 = openbsd.cs470.internal
```

… to define our subjectAltName for all requests issued from this system to be our system's fully-qualified DNS hostname.

One can rapidly see why it's so nice to have a public CA sign your organization's root certificate, and why they charge tens of thousands of dollars for the service; if it's signed by a common, public trust anchor, you don't need to copy your root CA certificate to all your systems, and tell every system and piece of software to trust your CA, as we'll have to do for every service using encryption through the rest of the labs.

For more quality reading on what you can do with an OpenSSL root certificate authority, including some of what `CA.pl` is doing in the background, check out …

https://jamielinux.com/docs/openssl-certificate-authority/

… if you screw anything up and need to revoke or re-issue a certificate, this site could come in handy.

part two: set up directory service (LDAP)

LDAP is short for lightweight directory access protocol, a more successful successor protocol to DAP, which was also known as X.500. For this reason, LDAP is often called "X.500 lite" … it was originally conceived as a way of accessing X.500 directories via TCP/IP, rather than with the OSI protocol stack, the same OSI model of the famous network layers 1 through 7.

When setting up large numbers of computing instances across a network, it's far more convenient to aim systems at a directory of user information, like LDAP. Microsoft's Active Directory is a value-added implementation of LDAP, with extensions for host configuration, the optional inclusion of DNS, system access control information, network topology, and Windows-based network security management. Many looked at Active Directory at the time of its inception as a classic case of "embrace, extend, and exterminate" business tactics, and they were not wrong … then. Like I've said before, though, that's a different company in Redmond today, and Microsoft has supported the free and open-source Samba project to support Active Directory and open its directory schema on other platforms.

Recommended reading:
https://en.wikipedia.org/wiki/Embrace,_extend,_and_extinguish

We want to set up, activate, authorize, and disable users in one place, not on each system, and LDAP provides us the ability to do that. Over time, technology teams have looked at and used various platforms to do this, notably NIS and NIS+, which were also used in prior iterations of this class. NIS was conceived before we matured in our thinking about security, and became unsafe to run over time as more and more security flaws were discovered in its design and implementations. NIS+ ("NIS Plus") served to mitigate most of the discovered risks and design flaws in NIS, including providing for authentication between servers, and encryption and replication of data, but was widely perceived as a pain in the ass, and nobody used it.

LDAP soon became the obvious replacement answer. By having a comprehensive directory in this service niche, the directory becomes authentication glue, a way of federating names internal to an organization, discovering services, providing for registration, removal, and fault tolerant replacement. It's a vehicle for providing identity, policy, and management as a service. It offers the ability to create a user in a single place when they join, and to disable a user in a single place, with every change made instantly spreading across the entity.

The name "directory" here isn't a tricky name; these directories are also supposed to support the same lookup functionality we want for a directory of employees or members, including phone numbers, physical mailing addresses, office numbers, e-mail addresses … and certificates for performing encryption key exchange between entities. The standards body that developed X.500 directories also developed the schema for the encryption certificates we use today with TLS (transport layer security), the general-purpose API used to add encryption and authentication to a lot of common protocols, notably HTTP, which we call HTTPS when used with TLS. You might have noticed "x509v3" mentioned in printing out certificates previously; the standards were obviously numbered similarly, and intended to be used together.

This means that LDAP and certificates share a common naming schema for entities (the distinguished name), so that the two types of data can be used together, and to refer to each other. For more reading, refer to: https://en.wikipedia.org/wiki/X.500#The_relationship_of_the_X.500_Directory_and_X.509v3_digital_certificates

!! IMPORTANT NOTE: failures in this part of the lab could break your ability to log into your VM. If things break, you may have to forcibly reset your VM (this is one of the few use cases where it's okay), reboot into single-user mode, and mount the filesystem read/write in order to check log files for error messages and repair configuration files. Make sure you always keep a copy of original configuration files so that you can revert back to a working configuration that'll boot into multi-user mode without hanging.

You may also want to take a few right now to investigate snapshotting your VM with VMware. I recommend you do a web search and read about what it means before you do it. You will want to clean up afterwards, once you're sure that everything works. Snapshots can eat up a lot of storage.

If you're using UTM, UTM doesn't yet support snapshotting. The underlying hypervisor technology (KVM/QEMU) does, but the UTM user interface on top of it doesn't implement it, hence the word "yet" in the prior sentence. Take a moment to set up backups or investigate making copies of your VM files. `tar` is a great quick 'n' dirty backup tool. Note that you'll want to shut down your VM before copying its file, so that you're not copying a working state of the VM … if you were to do that, trying to fire up a VM from the backup would be very much like starting up a Unix system after not shutting it down properly (discussed much in lab 1). Since nothing is counting on your OpenBSD VM, it'll be easy to shut it down now to back it up. That will be harder starting with lab 2, once another system requires your OpenBSD VM for DNS and authentication.

If you find yourself unable to log in, remember to try SSHing in as `failsafe` first. It's for this very reason that account is called "failsafe" … if we fail to get LDAP working, or if it breaks, `failsafe` is a local account that should hopefully work. OpenBSD also refuses to let root log in on console in multi-user mode, no matter what, even when you dumb down the terminal security settings we tweaked in lab 1. With key-based authentication and this account's special group memberships, you should be able to log in and cleanly reboot the VM. If that doesn't work, try telling it to reboot via VMware's controls for your VM. On the Mac, this will

be under the "Virtual Machine" menu. In VMware Workstation, this will be in the "VM" menu, "Power" sub-menu. Wait a minute or two before forcefully powering down or resetting the VM. **Again, when you reboot the VM, you'll probably need to use single-user mode to fix problems here.** Remember, in single user mode, you'll need to set the terminal type in the environment to use `vi` and will also need to mount the root filesystem read/write to change any files.

On all our VMs moving forward, you'll be setting up a local `failsafe` account to get LDAP working. After you get LDAP authentication working on each system, you should be using your LDAP user for everything else in each lab.

That said, let's get started.

10. OpenBSD includes its own built-in LDAP support, but it's only a server (`ldapd`), and we don't use it anymore because of some limitations we won't have with OpenLDAP. Let's install OpenLDAP to provide the missing pieces. The server is installed here, though not used later, to show you how to handle OpenBSD's port "flavors" concept.

    ```
    $ cd /usr/ports/*/openldap
    ```

    I use the star (remember, "asterisk" is too many syllables) here because it's handy to not have to know the name of an intermediate folder, or how a piece of software is categorized if you know exactly what you're looking for (OpenLDAP). If I'd done only that, though, I'd be missing the opportunity to make an important point: look where you're at.

    ```
    $ pwd
    /usr/ports/databases/openldap
    ```

    Don't be daunted by thinking this LDAP thing is any kind of crazy new thing … it's just a database, with its own particular syntax for storing and querying extensible data for various kinds of things: people (users), groups, computers, networks, and so on and so forth. Each entity can authenticate to the database and have various public (phone numbers, e-mail addresses, user ID numbers), and non-public fields (passwords).

    The OpenLDAP port builds both the server and client subpackages. I recommend you briefly check out the OpenBSD Ports FAQ for more information about port flavors and subpackages. You will need both on your OpenBSD VM as it is both the server and a client for authentication.

    ```
    $ make show=PKGNAMES
    openldap-client-2.6.8v0 debug-openldap-client-2.6.8v0 openldap-server-2.6.8v0
    debug-openldap-server-2.6.8v0
    $ make show=MULTI_PACKAGES
    -main -server
    ```

    Build and install the server package. Note: you should see this install the client too.

    ```
    $ sudo env SUBPACKAGE=-server make install
    ```

    In the command above, `env` is used to set an environment variable before running a command. Since `sudo` filters environment variables for security purposes, this can come in most handy.

```
===>  Installing openldap-client-2.6.8v0 from /usr/ports/packages/amd64/all/
openldap-client-2.6.8v0: ok
===> Returning to build of openldap-server-2.6.8v0
===> openldap-server-2.6.8v0 depends on: openldap-client-* -> openldap-client-
2.6.8v0
===>  Verifying specs: c crypto pthread sasl2 ssl lber ldap ltdl m perl sodium
===>  found c.100.3 crypto.55.0 pthread.27.1 sasl2.3.1 ssl.58.0 lber.16.0 ldap.16.0
ltdl.5.0 m.10.1 perl.24.0 sodium.10.2
===>  Installing openldap-server-2.6.8v0 from /usr/ports/packages/amd64/all/
openldap-server-2.6.8v0: ok
The following new rcscripts were installed: /etc/rc.d/slapd
See rcctl(8) for details.
```

You should see something like the above after a little while; note the last few lines and its comments about `rcctl` … you should remember how we managed BIND/`named` via its `rcctl` startup script and object name, `isc_named`. We can manage OpenLDAP's server process, `slapd`, the same way.

11. Check out the `man` page for `slapd` on OpenBSD. I always found it funny that some LDAP daemons call themselves `slapd` … this is somehow supposed to stand for "standalone LDAP daemon," but I'd picture a daemon serving up a swift whack across the face, slapstick style, like a Three Stooges as-a-service (3SaaS?).

    ```
    $ man slapd
    ```

    Please note the options for starting up `slapd` in debug mode, which might be useful later in troubleshooting, and **the URL for the OpenLDAP Administrator's Guide**. **Go to that URL**, and start skimming … we're going to be leaning on this documentation for the next few steps.

    No, really. Skim it. I made a full step out of this recommendation to check out the documentation for reason.

12. Add the new CA certificate to the trust anchors for OpenLDAP. Use `sudo` and `vi` to edit the file `/etc/openldap/ldap.conf`, and add this line to aim the LDAP client at our chain of trust, including our local certificate authority's root certificate.

    ```
    TLS_CACERT  /etc/ssl/cert.pem
    ```

While you're in there, you should also set the LDAP base to `dc=cs470,dc=internal` and set `URI` to `ldaps://openbsd.cs470.internal` … uncomment these two lines by removing the leading pound sign. `dc` is short for "domain component," and you will note here that we're defining our OpenBSD VM as the LDAP server for an authentication domain to be tied to our lab's DNS domain, cs470.internal.

13. We need to issue a certificate for our new LDAP server, using our new certificate authority, so first, let's generate a new key for our LDAP service. First, let's create a directory to contain the encryption materials for our LDAP service …

```
$ sudo mkdir /etc/openldap/ssl
```

… then let's use `openssl` to generate the key, and this time, let's just use a 3k-bit key to reduce the compute load associated with LDAP traffic just a little bit.

```
$ sudo openssl genrsa -out /etc/openldap/ssl/slapd.key 3072
Generating RSA private key, 3072 bit long modulus
.....................................................................
.....................................................
e is 65537 (0x010001)
```

Change the permissions on the key file to make it only readable by the owner and nobody else.

```
$ sudo chmod 400 /etc/openldap/ssl/slapd.key
```

14. Issue a certificate signing request (CSR) for the directory server.

```
$ sudo openssl req -new -key /etc/openldap/ssl/slapd.key -out
/etc/openldap/ssl/slapd.csr
```

Note how we pre-configured `openssl.cnf` has saved us time here … leave the locality/city name, organizational unit, and e-mail address fields blank, provide `openbsd.cs470.internal` for the common name to be associated with the new certificate and certificate request.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [Californistan]:
Locality Name (eg, city) []:
Organization Name (eg, company) [SDSU CS470]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:openbsd.cs470.internal
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

We could set up a certificate for each of multiple services on a system, and name the certificate for the service, using the "common name" field … this is why having the DNS name in subjectAltName is important. For the sake of simplicity here, though, we're just going to do one certificate per system.

You will become familiar with this dance very soon; as stated above, all services we typically provide a password for, we will want to provide encryption for, and that means a certificate. The challenge password for the CSR file is so that somebody can't find your CSR, take it to another CA, and get a certificate for your computer. Hopefully needless to say, that would be **bad**.

15. Now approve the certificate signing request. Copy the CSR out of the OpenLDAP configuration directory to make sure it's visible to the PKI, without becoming root …

```
$ cd /home/pki && sudo cp /etc/openldap/ssl/slapd.csr ./newreq.pem
```

… approve the certificate request by having the CA sign it …

```
$ CA.pl -sign
```

… you should see something like the following. Since you generated the CSR, just make sure the data matches your entry earlier in this step before answering `y` to sign the certificate, then commit it to the CA's databases.

```
====
openssl ca  -policy policy_anything -out newcert.pem -infiles newreq.pem
Using configuration from /etc/ssl/openssl.cnf
Enter pass phrase for /home/pki/data/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            e8:ce:43:c6:88:92:7f:1e
        Validity
            Not Before: Feb  8 22:22:42 2025 GMT
            Not After : Feb  8 22:22:42 2026 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = Californistan
            organizationName          = SDSU CS470
            commonName                = openbsd.cs470.internal
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                77:0A:67:A8:21:51:26:7A:08:99:4D:8C:B5:3B:A0:1E:9C:F4:6B:DC
            X509v3 Authority Key Identifier:
                keyid:AC:48:5B:D9:D0:89:76:E5:90:36:C0:C2:3B:FB:D2:AB:2B:FB:45:9E

            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Subject Alternative Name:
                DNS:openbsd.cs470.internal
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
Certificate is to be certified until Feb  8 22:22:42 2026 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
```

```
Write out database with 1 new entries Data Base Updated
==> 0
====
Signed certificate is in newcert.pem
```

Copy the certificate to OpenLDAP's certificate directory …

```
$ sudo cp newcert.pem /etc/openldap/ssl/slapd.crt
```

… and finally, clean up a bit. Let's make sure the new certificate for `slapd` is owned by the user _openldap (the underscore is a part of the username, and is meant to denote the user is a non-interactive owner of a subsystem) …

```
$ sudo chown -R _openldap:_openldap /etc/openldap/ssl
```

… and remove the certificate request and new certificate in `/home/pki` for the next go-around.

```
$ sudo rm /home/pki/newcert.pem /home/pki/newreq.pem
```

Even though root (or you using `sudo`) created the certificate request, you have write permissions for `/home/pki` and this means you're able to delete even a root-created file there. `rm` may confirm you really want to delete it, though.

16. Now we configure `slapd`. Strap in and read carefully … this one's a doozy.

    Just like we did in lab 1 with BIND, use `find` to discover the location of a sample `slapd.conf` under `/usr/local` and use `sudo` and `cp` to copy it to `/etc/openldap/slapd.conf`. After copying it over, `chown` it as in the previous step. Then use `sudo` and `vi` to edit the configuration file.

    The configuration file begins with including "schema," the naming conventions for the directory database's contents. `core.schema` is included, but that's not enough to store and authenticate POSIX users. Clone that line, and add lines including both `cosine.schema` and `nis.schema` from the same schema directory. The first has nothing to do with trigonometry. The second is definitely a reference to the historic NIS service we discussed earlier. While you're at it, take a quick look at both files.

    In the MDB section, change the database suffix to `dc=cs470,dc=internal` to match what we did with `/etc/openldap/ldap.conf` (the LDAP client configuration file), and change the administrative "distinguished name" in the `rootdn` field to `cn=admin,dc=cs470,dc=internal` …

    In `man slapd.conf`, read about setting up logging. Note that for some strange reason, OpenLDAP has chosen to make `slapd` always log with the "debug" severity, and that it logs by default with the "local4" facility. Please add `loglevel 256` to `slapd.conf`, near the top of the file, just above the comment declaring the location of the configuration database definitions.

    Create a password for the directory administrator. I often use a utility called `pwgen` (found in `/usr/ports/security` on OpenBSD) to create strong, random passwords. You can use whatever you want here for a password, however. As the config file implies, use the utility `slappasswd` to create a hash of your directory admin password; like passwords, we don't store them in the clear. Use no

arguments to use the default, strongest encryption available.

Insert your new directory admin password on the `rootpw` line under the `rootdn` configuration for the MDB dataset (**not** the other locations that are commented out) in place of the place holder "secret," **including** the portion in curly brackets that denotes the hashing algorithm used.

Finally, entries in the directory database are indexed by `objectClass` by default because we will often search for entries based upon their type. We'll also be searching for them based upon the username of a user logging in, so add in a second line to index by the attribute `uid` as well.

This configuration file now contains a password hash for our directory admin account … let's make sure nobody who doesn't need to do so is able to read it and try to crack that hash.

```
$ sudo chown _openldap:_openldap /etc/openldap/slapd.conf
$ sudo chmod 640 /etc/openldap/slapd.conf
```

17. In order to make sure `syslogd` appropriately saves logs we just configured in `slapd.conf`, use `sudo` and `vi` to edit `/etc/syslog.conf`. Underneath the line that sends info-level daemon logs to `/var/log/daemon`, add a second line that sends `local4.debug` also to `/var/log/daemon`. Send `syslogd` a HUP ("hang up") signal to tell it to restart and re-read its configuration file with …

```
$ sudo pkill -HUP syslogd
```

Worth noting: this command is functionally equivalent to `sudo rcctl restart syslogd`.

Also worth noting: most BSDs implement a keyword-based multi-process version of `kill` as `killall`, but OpenBSD calls its version `pkill` like commercial Unix does, a strange and seemingly out-of-character decision.

18. Now enable and start `slapd` just like we did with `named` before it.

```
$ sudo rcctl enable slapd
$ sudo rcctl start slapd
```

If you don't receive an `ok` back, look through logs in `/var/log/daemon` … or the rest of `/var/log` … or start up `slapd` in debug mode, and find what you missed. Make sure that logs for `slapd` are showing up in `/var/log/daemon`, or we'll lose visibility of what's going on inside `slapd` if we have issues later.

You can also confirm it's running with `ps` …

```
$ ps auxww | grep ldap
_openlda 26462  0.0  0.2 17812  4572 ??  S       7:42PM    0:00.06
/usr/local/libexec/slapd -u _openldap
failsafe  3985  0.0  0.0  2204   904 p0  R+/0    7:42PM    0:00.00 grep ldap
```

… and `netstat`.

```
$ netstat -a | grep ldap
tcp          0      0  *.ldap                 *.*                    LISTEN
```

```
tcp6       0      0  *.ldap                 *.*                    LISTEN
```

Note that we're not using the `-n` switch here because we wanted to search for `ldap` … it might appear on a couple of ports … this is okay.

```
$ grep ldap /etc/services
ldap            389/tcp                         # Lightweight Directory Access
ldap            389/udp
ldaps           636/tcp                         # LDAP over TLS/SSL
ldaps           636/udp
```

19. Of course, we issued our LDAP service a certificate to get LDAP over TLS, with encryption, and because you only see `ldap` in `netstat` output above, that means we're only running an *unencrypted* version of the LDAP service. Since we're going to send sensitive data to this service – passwords used for authentication – this simply won't be acceptable.

    Just like we did in lab 1 with `named`, use `ps` to figure out the default switches for `slapd`, and add a line to `/etc/rc.conf.local` to tell `slapd` to start with its default settings **plus** the settings to listen on LDAPS (LDAP with TLS for encryption) and LDAPI (a socket interface on the filesystem, at `/var/run/ldapi`). You are going to need to read through `man slapd` to find the appropriate configuration arguments. We do not want LDAP, just LDAPS and LDAPI. You'll find directions in `man slapd` … check out the `-h` switch. Also note that you can use `rcctl` to check defaults, and get and set actives settings for services … see `man rcctl` … it is not a long read.

    Back in the OpenLDAP 2.6 Administrator's Guide, pull up the page for server-side TLS configuration …

    https://www.openldap.org/doc/admin26/tls.html#Server%20Configuration

    … as it says, in the "global directives section" (anywhere before any of the database sections in `slapd.conf`), aim `slapd` at the monolithic CA certificates file (the "trust anchors" file) in `/etc/ssl`, the certificate you just created for your LDAP server (`slapd.crt`), and the corresponding private key file (`slapd.key`). It needs to be aimed at all three to set up TLS: its certificate, its key file, and the trust anchor for its certificate.

    Finally, use `rcctl` to `stop` and then `start` the LDAP service. The output of the same `netstat` command above should now show references to `ldaps` and `/var/run/ldapi` afterwards if you got it correctly, **not** to `ldap`.

    Since TLS just provides an encryption later for use with arbitrary sockets, OpenSSL provides a tool to help troubleshoot TLS connections. You should be able to use it now to connect to port 636 on your OpenBSD machine to validate that you correctly configured the certificate on the LDAP server, and correctly configured the trust relationship with your new root CA in your TLS client.

    ```
    $ openssl s_client -showcerts -connect openbsd.cs470.internal:636
    ```

    Even though we're (probably) connecting to the LDAPS service just by means of hostname and port number, this creates far more trust, having a digitally-signed certificate attest to this information. Again, if we wanted, we could call it "cs470.internal directory server," since the subjectAltName field

attests to trust using the port number and DNS hostname.

```
CONNECTED(00000003)
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
---
Certificate chain
 0 s:C=US, ST=Californistan, O=SDSU CS470, CN=openbsd.cs470.internal
   i:C=US, ST=Californistan, O=SDSU CS470, CN=CS470 2025 Spring CA
   a:PKEY: rsaEncryption, 3072 (bit); sigalg: RSA-SHA256
   v:NotBefore: Feb  8 22:22:42 2025 GMT; NotAfter: Feb  8 22:22:42 2026 GMT
-----BEGIN CERTIFICATE-----
…
-----END CERTIFICATE-----
 1 s:C=US, ST=Californistan, O=SDSU CS470, CN=CS470 2025 Spring CA
   i:C=US, ST=Californistan, O=SDSU CS470, CN=CS470 2025 Spring CA
   a:PKEY: rsaEncryption, 4096 (bit); sigalg: RSA-SHA256
   v:NotBefore: Feb  8 20:51:44 2025 GMT; NotAfter: Feb  8 20:51:44 2028 GMT-----
BEGIN CERTIFICATE-----
…
-----END CERTIFICATE-----
---
Server certificate
subject=/C=US/ST=Californistan/O=SDSU CS470/CN=openbsd.cs470.internal
issuer=/C=US/ST=Californistan/O=SDSU CS470/CN=CS470 2025 Spring CA
---
No client certificate CA names sent
Server Temp Key: ECDH, X25519, 253 bits
---
SSL handshake has read 3482 bytes and written 388 bytes
---
New, TLSv1/SSLv3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 3072 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID:
    Session-ID-ctx:
    Master-Key:
    Start Time: 1739057512
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)---
^C
```

As you can see, I hit `control-c` to end the tool; after the initial connection, `s_client` allows you to piggyback on the established encrypted connection … you could pipe data into this command to send it to the server at the other side of the connection, or manually interact with it using keyboard (standard input) … if that were feasible.

Before it does that, though, it gives us this output, showing how `s_client` authenticates the server's certificate. First it authenticates the root CA certificate, which it successfully validates by comparison with the trust anchor database under `/etc/ssl`. Then it validates the server certificate, which it does cryptographically, using the public key from the root certificate. If `s_client` isn't able to establish a connection or validate the certificate, you would receive a valuable error message here, telling you

what went wrong validating the certificate.

20. Initialize the new directory with the "base" of the LDAP tree. When importing and exporting from an LDAP directory, contents are stored in a format called LDIF (LDAP data interchange format). `cd` into your home directory and create a file called `base.ldif` with the following contents:

```
# cs470.internal
dn: dc=cs470,dc=internal
objectClass: top
objectClass: dcObject
objectClass: organization
o: CS470
dc: cs470

# users, cs470.internal
dn: ou=users,dc=cs470,dc=internal
objectClass: organizationalUnit
ou: users

# groups, cs470.internal
dn: ou=groups,dc=cs470,dc=internal
objectClass: organizationalUnit
ou: groups
```

This serves, literally, as the top of the hierarchy; we have created the organization itself, CS470, and the first "organizational units" (`ou`) within the directory, to provide homes for user accounts and groups in the directory.

Import the LDIF into the directory with the following command:

```
$ ldapadd -x -W -D cn=admin,dc=cs470,dc=internal -f ./base.ldif
```

Note that `-D` here specifies which user you're logging in as, and you'll be prompted for the directory admin password you created earlier. If everything was done correctly, you should see the following output:

```
adding new entry "dc=cs470,dc=internal"

adding new entry "ou=users,dc=cs470,dc=internal"

adding new entry "ou=groups,dc=cs470,dc=internal"
```

If you're having problems, please read `man slapd` and `man ldapadd` … and I'll give you a few helpful hints here too.

First, always, always look at all your logs. In `/var/log` on OpenBSD, this means to look at the logfiles `authlog` and `daemon` and `messages` and `secure`. All are potentially pertinent. The `-r` switch with `grep` allows you to easily recursively search a whole directory tree.

Also, the OpenLDAP client tools all have a `-v` switch you can use to make them give you more output.

If you simply "can't connect," it's either a certificate issue or you've got the client configuration messed up. When it's a login issue, you're probably going to be told "failed login" or "failed bind."

Finally, the `-d` switch from the `man` page runs the LDAP server in debug mode, with output to your terminal. If you're having problems getting LDAP to work correctly, run `slapd` in debug mode in one SSH session, and the `ldapadd` commands in another. See what pops up; it'll probably clue you into what you missed above.

21. Now let's add a user account, but we're going to add it to LDAP, **not** to the local user databases. Use whatever name you like here, but pick one you like and can stick to. It's going to be really handy if it matches the username used on your lab computer's host operating system.

    I do **not** recommend that you use any capital letters in your user or group names; users and groups are lower case by convention. Lots of software will not handle upper-case gracefully.

    I also recommend that you pick a short one; some operating systems still have a stale preference for eight character or less usernames, and you'll be saving yourself some minor headaches later on in labs 5 and 5b.

    By having user authentication handled in a directory, you'll be using the same username on all your VMs, and it'll be better still if it's the same username as you use in the Unix/Linux environment on your host … it'll save you from having to specify a username with `ssh` and `scp` commands. If nothing pops into mind immediately, I recommend just using your first name here.

    Create a password for your new user, and encrypt and save it as we did above using `slappasswd`. Go to `~failsafe` and create a file called `user.ldif` with the following contents … use UID and GID 1001, but replace the contents of `cn`, `uid` (in the `dn` too), `homeDirectory`, `gecos`, and `userPassword`.

```
dn: cn=peter,ou=users,dc=cs470,dc=internal
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
cn: peter
uid: peter
uidNumber: 1001
gidNumber: 1001
homeDirectory: /home/peter
loginShell: /usr/local/bin/bash
gecos: Peter Bartoli
shadowMin: 0
shadowMax: 0
shadowWarning: 0
userPassword: {SSHA}redacted
```

    Note that because we're using `/usr/local/bin/bash` here, we're going to need to make sure that path works on every other VM we set up moving forward.

    Add the user account with the following command:

```
$ ldapadd -x -W -D cn=admin,dc=cs470,dc=internal -f ./user.ldif
```

    After providing your admin password, you should see something like this as output:

```
adding new entry "cn=peter,ou=users,dc=cs470,dc=internal"
```

22. Create a third file, `groups.ldif`, with some more groups to use with our VMs. We're going to create a group for permissions to our CA files here on our OpenBSD VM, and a group to unify `sudo` configuration for all our VMs.

```
dn: cn=peter,ou=groups,dc=cs470,dc=internal
objectClass: posixGroup
gidNumber: 1001
cn: peter
memberuid: peter

dn: cn=sudoers,ou=groups,dc=cs470,dc=internal
objectClass: posixGroup
gidNumber: 420
cn: sudoers
memberuid: peter
memberuid: failsafe
```

Add the groups to the directory with the following command:

```
$ ldapadd -x -W -D cn=admin,dc=cs470,dc=internal -f ./groups.ldif
```

After providing your admin password, you should see something like this as output:

```
adding new entry "cn=peter,ou=groups,dc=cs470,dc=internal"

adding new entry "cn=sudoers,ou=groups,dc=cs470,dc=internal"
```

23. Make your new user account's home directory, and give it ownership and exclusive group access to its files.

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

Of course, replace my username with yours. You'll get used to this and I'll stop saying it before too long passes by. Look at this directory with `ls -l` and note that user and group number 1001 are not yet mapped to a name; we'll fix this shortly.

24. Before setting up login over LDAP, it'll be useful to test your directory setup a bit.

Search for your user …

```
$ ldapsearch -x "(objectclass=posixAccount)"
```

… you should see the entire record you created above in the output. Note how we are not authenticating here … changing the directory requires authentication, but by default, like a true directory, and like `/etc/passwd` and `/etc/group`, some information is considered public and doesn't require authentication to ask for. We could, of course, configure the directory to require authentication for all searches, but this isn't necessary on a private network like our VM lab nets.

Also, search for your groups.

```
$ ldapsearch -x "(objectclass=posixGroup)"
```

Note that the second argument to `ldapsearch` is the search query, and you can add another mandatory search token with an ampersand (`&`) to specify a logical AND operation inside another set of parenthesis, like this …

```
$ ldapsearch -x "(&(objectclass=posixGroup)(cn=peter))"
```

… or a logical OR with the vertical bar or pipe character …

```
$ ldapsearch -x "(|(objectclass=posixGroup)(objectclass=posixAccount))"
```

… note how LDAP uses prefix notation, in its own way, with the operator coming before the two operands. This last query should spit out your user account and the two groups you just created.

If you're having problems, please read `man slapd` and `man ldapsearch` and be sure to consult your logs.

25. Time to set up login over LDAP. Use `sudo` and `vi` to create `/etc/login_ldap.conf` with the following contents.

```
host=ldaps://openbsd.cs470.internal
basedn=ou=users,dc=cs470,dc=internal
scope=sub
filter=(&(objectclass=posixAccount)(uid=%u))
```

Don't be fooled by the monospaced font. The LDAP search filter above has no spaces in it. In fact, none of the lines above have any spaces in them.

Our search filter requires a matching LDAP object to be a POSIX account and to match the username that is attempting to log in.

Because this file becomes key to authentication and because we're on OpenBSD, we're going to have to set the permissions accordingly or it will complain about the config file's permissions.

```
$ sudo chown root:auth /etc/login_ldap.conf && sudo chmod 640 /etc/login_ldap.conf
```

26. Now let's test login over LDAP before we add it to our system's login configuration … we can do this by means of directly calling the tool OpenBSD uses to log a user in over LDAP.

```
$ sudo /usr/libexec/auth/login_ldap -d -s login peter
```

Note that the first password prompt might be for `sudo`, for your failsafe user, and that you may get prompted for a password twice if you haven't used `sudo` in a while.

My results looked like this, and yours will too, if you got the setup correctly.

```
Parsing config file '/etc/login_ldap.conf'
parse_server_line buf = ldaps://openbsd.cs470.internal
```

```
host openbsd.cs470.internal, port 636
connect success!
do_conn: starting TLS
bind success!
0: search (ou=users,dc=cs470,dc=internal, (&(objectclass=posixAccount)(uid=peter)))
1: msgid 2, type 04
1: SEARCH_ENTRY userdn cn=peter,ou=users,dc=cs470,dc=internal
1: msgid 2, type 05
1: returning userdn = cn=peter,ou=users,dc=cs470,dc=internal
userdn cn=peter,ou=users,dc=cs470,dc=internal
user bind success!
authorize
```

You should be able to generally understand the output of this command at this point in time, and what it's saying if it fails.

27. Now, aim OpenBSD at LDAP for user accounts, and group membership information.

    OpenBSD does this in a rather novel way, by having LDAP behave as a backend for the old Network Information Service (NIS) mentioned earlier. NIS was initially to be called the "Yellow Pages" … and in this case, AT&T/Bell owned the trademark and objected, so the name of the service was changed, but it kept `yp` as a part of its command and filesystem nomenclature.

    ```
    $ sudo cp -p /etc/examples/ypldap.conf /etc/ypldap.conf
    ```

    Edit the new config file. Fix all instances of the domain name, of course, and put in the correct `binddn`, but comment it and `bindcred` out … we don't actually need to log into LDAP to search the directory. Change 127.0.0.1 to `openbsd.cs470.internal` and add the keyword `ldaps` right after it – but before the curly bracket – to make sure it uses LDAP over TLS/SSL. Make the `fixed attribute class "ldap"` … and then in the first configuration block, declare that we're doing a bind to a local LDAP server (on this host) with `bind local`. Then, save out the file.

    Questions? See `man ypldap` and `man ypldap.conf`. Especially, note the switch in the `man` page for `ypldap` for testing your configuration file. **Do that now.** Also note the switches for debug mode and verbose mode.

28. Now, some NIS configuration. Set the NIS domain name immediately with this command.

    ```
    $ sudo domainname cs470.internal
    ```

    The next few commands add to root-owned files, so for convenience, let's run a root shell temporarily to make sure output redirection is done as root.

    ```
    $ sudo bash
    ```

    Configure cs470.internal as the NIS domain name after subsequent reboots. Note that with a single `>` we're truncating this file, if it already exists.

    ```
    # echo 'cs470.internal' > /etc/defaultdomain
    ```

Now tell the password and group databases to also refer to NIS. With `>>` we're appending to the ends of these files, not erasing any existing contents.

!! IMPORTANT: make sure you get both greater-than symbols here, or you'll wipe out your password and/or group databases, and that'll be hard to recover from. Maybe back them up to `.orig` files first?

```
# echo '+:*:::::::::' >> /etc/master.passwd
# echo '+:*::' >> /etc/group
```

Exit the root shell, and rebuild the password database so it refers to NIS.

```
$ sudo pwd_mkdb -p /etc/master.passwd
```

NIS clients typically used broadcast traffic to find its servers; we will short-circuit this and manually provide the server our NIS client is to bind to, also on our VM. Create the directory for `yp` server configuration …

```
$ sudo mkdir /etc/yp
```

… then create a file to aim the NIS client at the server on the same VM ...

```
$ sudo vi /etc/yp/cs470.internal
```

… into that file, enter only `openbsd.cs470.internal` and save it.

Finally, in case any services that require the name "openbsd.cs470.internal" to resolve start before the name server that provides it starts (which is also on this system), add it to `/etc/hosts`. `/etc/hosts` provides local overrides for name resolution. `sudo vi /etc/hosts` and add `openbsd` and `openbsd.cs470.internal` separated by spaces after localhost as alternate hostnames for 127.0.0.1, the loopback adapter.

29. Here's the moment of truth, really. If your OpenBSD VM is going to freak out, **this** is where it's likely to happen. If OpenBSD thinks it's got a backend behind NIS, but NIS times out, every process that does a user or group name lookup is going to hang.

    I'm not telling you to test `id` here quite yet; I'm showing you what failure looks like here. As always, read the whole step first, do it second. You might see something like this …

    ```
    $ id peter
    ^C
    $ sudo grep -ri ldap /var/log
    ^C^C^C^C^C^C^C^C^C^C^C^Z^Z^Z^Z^Z^Z^Z
    yp_match: clnt_call: RPC: Timed out
    yp_match: clnt_call: RPC: Timed out
    yp_match: clnt_call: RPC: Timed out
    ```

    The initial `control-c` to send a break to `id` during my first round of testing was able to get a command line back, but no such luck after I ran `sudo`. Even `control-z`, the control character to suspend a process and return a command line, didn't work. It just hung. I couldn't log in, couldn't SSH in … you get the idea.

if you see the above, this is when it's going to **have** to be okay (it's really never okay) to give your VM a hard reset, to go back into single-user mode, so that you can read logs and/or to disable `ypldap` in `/etc/rc.conf.local` so the system will boot up without locking up.

Make sure you have two SSH sessions into your computer before you start testing `ypldap`. In one of them, become root with `su` or start a shell with `sudo`. In the other, use `sudo` to manually fire up `ypldap` manually in verbose mode **and** debugging mode (read `man ypldap`), so that it'll tell you what's going on … while you test, test, test! Note: we are using NIS' password and group "maps" here for testing, because `ypldap` is mocking up the data in LDAP to look like NIS. If it looks like NIS to OpenBSD, then our users and groups in LDAP will show up along with the rest in our local databases.

```
$ ypmatch peter passwd.byname
```

You should see something like …

```
peter:*:1001:1001:ldap:0:0:peter:/home/peter:/usr/local/bin/bash
```

… or you need to get busy looking at logs and troubleshooting. `id` should also show your new user.

```
$ id peter
uid=1001(peter) gid=1001(peter) groups=1001(peter), 420(sudoers)
```

Test the groups too.

```
$ ypmatch peter group.byname
```

You should see something like …

```
peter:*:1001:peter
```

… and this one …

```
$ ypmatch sudoers group.byname
```

… should produce something like this.

```
sudoers:*:420:peter,failsafe
```

If you `ls -l /home`, `ls` should now be able to fill in all the user and group names as well.

Remember, you have to re-start services after changing their configuration files … they don't just magically watch all those config files, nor do we want them to do so! In debug mode, this is just a matter of hitting `control-c` and restarting the daemon manually after making the appropriate edits. When a daemon is running normally, you often have to fully stop the daemon with a utility like `rcctl` or its startup script, as issuing a restart may not tell a daemon to re-read its configuration file … it depends on the daemon.

Once you're happy `ypldap` works properly, enable it so that it starts up with your OpenBSD VM, and start it up normally now.

```
$ sudo rcctl enable ypldap
$ sudo rcctl start ypldap
```

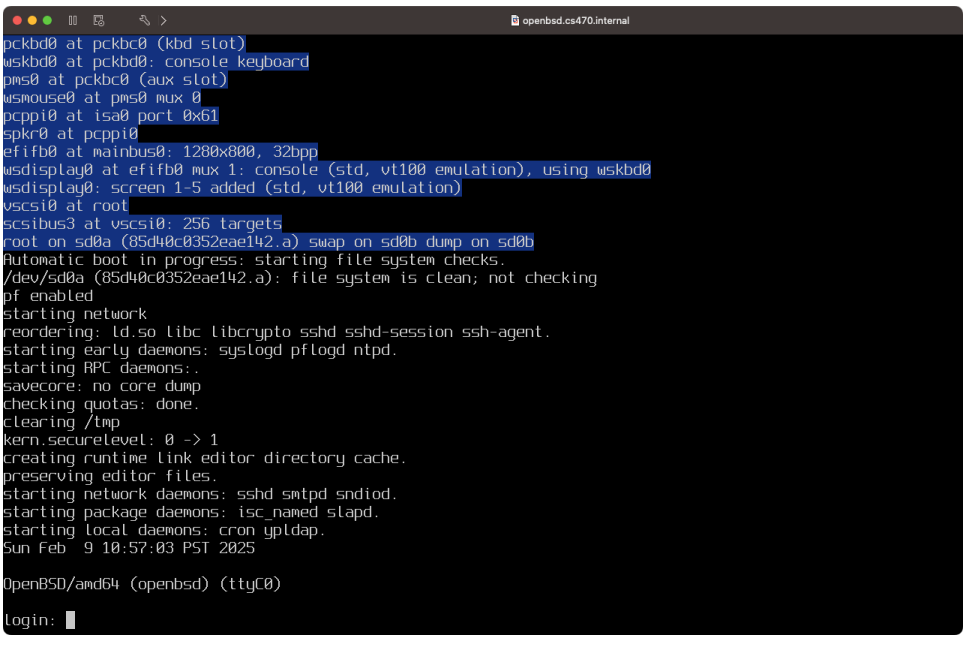
30. If `ypldap` starts before `slapd`, or the two can't talk for any other reason, our OpenBSD VM will lock up. Let's alter OpenBSD's service start-up order to fix this.

    Let's back up `/etc/rc` … it's OpenBSD's monolithic startup script, a very modernized relic of the old days of Unix. Not many operating systems still have an `/etc/rc` in that place, in that role, but OpenBSD does. If we mess up our edits, no matter how trivial, we want a way back.

    ```
    $ sudo cp -p /etc/rc /etc/rc.orig
    ```

    Now, use `sudo` and `vi` to edit `/etc/rc` ... find where OpenBSD starts `ypldap` with the `start_daemon` function, and remove `ypldap` as an argument there … leave the rest. As you might notice, a couple lines above in the section about "early daemons," OpenBSD starts up its own LDAP server (`ldapd`) early in the boot process if it's enabled. Starting `slapd` early was the first thing I tried, but `slapd` wouldn't start up that early in the boot process. Find where "local daemons" are started up, near the end of the boot process, and add `ypldap` to the end of the `start_daemon` line there in that section of `/etc/rc` and save out the file.

    Test your configuration by rebooting. Success here means that none of your services hang or fail during startup, and your OpenBSD VM starts up able to resolve users with `ypmatch`, `ls`, and `id`, as a couple steps ago.



31. Now that LDAP is working and tested and your users groups in your directory are exposed to the operating system, add LDAP as an acceptable login facility. This is where we really stand to break

things, so first, back up `login.conf`. It'll really be good to be able to recover this file if it gets messed up.

```
$ sudo cp -p /etc/login.conf /etc/login.conf.orig
```

Edit the file, and add these lines just above `auth-defaults` and its comment …

```
ldap:\
        :auth=ldap:
```

… make sure to use a tab to indent the second line, like in the rest of the file. Then add `ldap` to the middle of the list of default authentication methods, after `passwd`. `passwd` means all the default local user directory files (`/etc/passwd`, `/etc/master.passwd`, `/etc/group` …), and we want users there (such as failsafe) to take precedence, even if LDAP is broken. This, again, is the whole point of the failsafe user. Note: this means that if a user has the same name locally as a user in the directory … you will never be able to log in as the user in the directory.

```
auth-defaults:auth=passwd,ldap,skey:
```

Try logging in as the user you created in LDAP, now, on the console of the VM (in the window inside VMware Fusion/Workstation), and over SSH. If it doesn't work, hit your logs and hit them hard.

32. Create the `.ssh` folder in your new user account's home directory with the appropriate permissions and copy your SSH public key over into the file `authorized_keys` in this folder.

    Also, copy your failsafe account's `~/.ssh/config` from lab 1 into your LDAP user's home directory. Make sure it works, and that you're able to SSH from your host, into your LDAP account on your OpenBSD VM, then into your EDORAS account, all without a password.

    Once that works, it's a great time to replicate any shell/prompt customizations into your LDAP account.

33. Make sure your regular user can flex admin mojo; add your LDAP user account to the operator and _shutdown groups in `/etc/group`. Users on each group definition line are separated by a comma. Do not add your LDAP user account to the wheel group.

    We're doing this in `/etc/group` to grant permissions associated with these groups to your LDAP user locally on your OpenBSD VM. Other operating systems will also often have different groups on top of the same numeric group IDs.

    Use `visudo` to create another entry in `/etc/sudoers` to allow members of the sudoers group to have `sudo` rights. Just like `vipasswd` is used to try to prevent conflicting or syntactically incorrect edits to `/etc/passwd`, `visudo` does the same for the `sudo` configuration file. A syntax error in the password database could prevent logins and possibly force a hard reboot to single-user mode to fix. A syntax error to the `sudo` configuration file could also force a single-user mode fix, as it's possible none of the rightful admins could become root as needed.

Make sure that your LDAP user can use `sudo` successfully.

34. Create a local group called `pki` in `/etc/group` with group number 509, and add the failsafe account and your LDAP user to that group.

    Then, `sudo chown -R failsafe:pki /home/pki` and `sudo chmod -R g+w /home/pki` to give both your failsafe and LDAP users rights to access and write to the `pki` directory, since you're going to be using administrative rights in both of these accounts at times.

    Make sure that this just gave your LDAP user access to the contents of `/home/pki`.

!! IMPORTANT REMINDER: The failsafe account on each system is just for getting LDAP set up … and having a local account to log in with in case LDAP ever fails. After you get LDAP login working on each VM if each lab moving forward, you should not be using the failsafe account for anything unless explicitly told to do so, or if an obvious solution requires it. Once you can use your LDAP user on each VM, use it for everything.

part three: tuning kernel and memory

In this addition to the lab, we'll reduce the resources consumed by our OpenBSD VM, both by the kernel inside the VM and by the VM within our lab computers. As stated early on in lab 1, OpenBSD is often used in network and network security gear, and OpenBSD is desirable here for its security mindset, built-in encryption support, its small footprint, and the easy customizability of the BSD kernel.

In this part of the lab, we're going to focus on the latter two: how small we can make a functional OpenBSD system, and how little we really need to know to try to accomplish this. The `GENERIC` kernel doesn't have a tricky name; it's exactly what it claims to be: a generic kernel with support for most devices you'd find in a common specimen of that architecture, for a generic-purpose installation.

As such, the `GENERIC` kernel includes lots of drivers and functionality we don't need inside each instance, and since all of that is in kernel memory, it's a part of every process' virtual memory space and never gets swapped out. It's the very definition of wasted memory: code that won't ever be run on our hardware, occupying memory that can't be used by anything else. We're going to build a kernel without all that. As you'll see later when we build a Linux kernel from source in lab 4, Linux uses a modular kernel to reduce a lot of the waste, but building something as small as possible is much harder to do with Linux.

In way older versions of this class, we used to use SunOS 4.x, and even though we didn't have the source for the kernel, we had compiled object files for each source file and the kernel still had a similar configuration file we could use to determine which objects were used to assemble a kernel.

35. This is often a process of trial and error; as you'll see, remove too much functionality or a device that you need, and your kernel may not boot. Because of this, we'll want to make a copy of your kernel from the end of lab 1 outside the file paths touched by the kernel installation process.

    ```
    $ sudo cp -p /bsd /bsd.cs470
    ```

This way, if the kernel at the default path doesn't boot correctly, you can always reboot into another known-good kernel. Remember, the boot loader prompt that you get right after the firmware splash at each power-up and reboot of your VM can not only be used to boot into single-user mode, it can also be used to boot an arbitrary kernel …

```
>> OpenBSD/amd64 BOOTX64 3.65
boot> b bsd.cs470
```

… `b` is short for "boot," and the kernel filename can always be trailed by whatever kernel options, like single-user mode, fit your use case for the current boot cycle.

I recommend you test rebooting into an alternate kernel file before your ability to keep working depends on it.

36. You've seen the OpenBSD kernel configuration file and how to use it to build the kernel in lab 1. cd into the appropriate kernel configuration directory for your architecture, either `/sys/arch/amd64/conf` or `/sys/arch/arm64/conf`. Use `cat` to get everything from `/sys/conf/GENERIC` and the architecture-specific `GENERIC` kernel configuration file into a single, monolithic file called `CS470.MINIMAL` …

```
$ cat ./GENERIC /sys/conf/GENERIC | sudo tee ./CS470.MINIMAL
```

… if you got a `GENERIC.MP` kernel out of the box, include the `GENERIC.MP` config files to make sure your process supports both cores …

```
$ cat ./GENERIC /sys/conf/GENERIC ./GENERIC.MP | sudo tee ./CS470.MINIMAL
```

… multiprocessor support will be a requirement for receiving credit for this step on both architectures.

Make sure to remove any `include` statements in the newly created kernel config file that reference the original configs. In lab 1, we made copies of each config file for our custom kernel and kept the include statements for simplicity's sake. However, since we just used `tee` to combine them into a single monolithic file, the `include` statements just import a bunch of duplicate configurations that will cause errors.

In order to figure out which lines of kernel configuration are unnecessary, I recommend you send a copy of `dmesg` output to your home directory. Your final minimal kernel configuration file will only need lines for each device that occurs here as they discovered on your system. Instances of the same devices on (`at` in the kernel configuration file and `dmesg` output) different bus types than the one you're currently using for that device are generally not required, and can save space.

Also, certain `option` directives will also be unnecessary; you will need to do research in order to figure out which. Some trial and error may be involved as well; some kernels will fail to build with too many options removed … and some will fail to boot.

Note that re-configuration of your kernel will need to be done in the appropriate `conf` directory, and that repeated builds of the kernel don't necessarily require you to clean your `compile` directory. You

may just need to `make` again.

Note that you don't have to finish this before starting lab 2 – this part of this lab isn't on the critical path I advertised at the front – but it will get more painful to reboot your OpenBSD instance after more VMs are counting on it being up later on.

37. Once you've got a working minimal kernel, use top to see how much RAM your VM is using when fully booted, running `named` and `slapd`. Adjust your RAM downward. 256 MB of RAM might seem pretty tiny, but I recently ran an OpenBSD DHCP+DNS server VM with only 64 MB of RAM.

    We'll be curious to see how little RAM some are able to fit their VMs in.

Extra credit will be awarded to the interns able to get the smallest **working** kernel running on both the AMD64 and the ARM64 architectures. In order to receive extra credit, you will have to demonstrate that your minimal kernel boots and runs a complete system, including all required services, during lab time in class. A kernel size leaderboard will be maintained on the shadow website, linked to here.

That's it for lab 1b. Congratulations; you're definitely past the two heaviest lifts in this class, and all of the material will look way easier with these last two labs in your rear-view mirror.

`</lab1b>`

appendix: more LDAP

This section is provided for your reference; you do not need to do this stuff, just observe in case you need it. If you need to troubleshoot LDAP, this will come in handy.

OpenLDAP, out of the box, expects you to use its command line tools to manage the contents of the directory. In fact, current versions of OpenLDAP prefer that you import a lot of the configuration, including schema, that we put into `slapd.conf` into the "config" database referenced in that file … using the command line OpenLDAP tools.

There are, however, lots of third-party tools for managing LDAP instances, including `shelldap` (an LDAP shell), `ldapvi` (a `vi` front-end for editing directory databases), and there are several web-based tools that you can plug into an LDAP instance to provide a more friendly interface for heathens. Run `ls -l /usr/ports/*/*ldap*` to see a bunch of tools you can experiment with, if you'd like some more LDAP to chew on.

This command will show metadata, and shows that OpenLDAP honors the "root DSE" search that caused our lab 5 VM to fail to use LDAP in prior years.

```
$ ldapsearch -H ldaps://openbsd.cs470.internal -x -s base -b "dc=cs470,dc=internal" -LLL
"+"
```

This is how you change a user's password in LDAP:

```
# ldappasswd -s NewPassword -W -D "cn=admin,dc=cs470,dc=internal" -x
"cn=peter,ou=users,dc=cs470,dc=internal"
```

To add a user to a group, create an LDIF file with a name like `groupmod.ldif` …

```
dn: cn=sudoers,ou=groups,dc=cs470,dc=internal
changetype: modify
add: memberuid
memberuid: peter
```

… or to remove a user …

```
dn: cn=sudoers,ou=groups,dc=cs470,dc=internal
changetype: modify
delete: memberuid
memberuid: peter
```

… then:

```
$ ldapmodify -x -W -D "cn=admin,dc=cs470,dc=internal" -f groupmod.ldif
```

To change an attribute, like a user's shell, as I will personally do to remain a Unix dinosaur but still recommend you don't, create an LDIF …

```
dn: cn=peter,ou=users,dc=cs470,dc=internal
changetype: modify
replace: loginShell
loginShell: /usr/local/bin/tcsh
```

… then run the same `ldapmodify` command as above.