Installing Pubcookie on Redhat AS3.0 and authenticating against Windows Active Directory

Table of Contents

Introduction

Pubcookie and Windows Active Directory are a good choice of tools when wanting to provide single sign on within an institute. Many institutions will already be using the Active Directory for user authentication and Pubcookie provides a robust and flexible layer on top to provide web-based secure single sign on. Pubcookie authentication can be used for Apache httpd 1.3.* and 2.0.* authentication, and for authenticating Microsoft IIS. It is also possible to use pubcookie with Zope, Tomcat and other technologies by passing credentials from Apache httpd.

Purpose

The purpose of this document is to lower the skill set required to install the Pubcookie Web based Single Sign on system and using it to authenticate against a Windows 2003 Active Directory. The guide should also work for those installing against Windows 2000 Active Directory. It will discuss the two authentication flavours that can be used against the Active Directory. It will also attempt to provide future proofing of the Kerberos flavour, in that it will allow easy upgrade to Windows 2003 without encountering problems relating to older Kerberos clients (as comes with Redhat AS3.0) and Windows 2003.

It is not the intention of this guide to give a full overview of how to install Redhat AS3.0. The Redhat installation documentation is good and should be used. However this document will outline the some of the Redhat installation steps that will enable easier setup of Pubcookie. This guide will focus on administration from the command line as most web servers run "headless" (i.e. No monitor, no windows, no graphical interface) so are only administrable by command line. Command line is available in all graphical environments.

About Pubcookie

Pubcookie is a WebISO, see http://middleware.internet2.edu/webiso/ for a full definition of WebISOs and for other WebISOs available. More information can be obtained about Pubcookie at http://www.pubcookie.org/.

Skill Sets required

In order to Install and manage a Pubcookie server you will need to be able to access the following skills

1. A reasonable working knowledge of the Linux (or Unix) Command Line Interface (CLI);
2. Knowledge of how to use the Apache web server, either the 2.0.* or 1.3.* versions;
3. Familiarity with the concepts of https communication (certificates, keys, etc.);
4. Familiarity with firewalls or access to someone who is familiar, in particular with Linux iptables style firewalls;
5. Familiarity with the setup of Windows Active Directory, or access to someone who has those skills;
6. Most importantly a willingness to read around subject areas, man pages, Google, and mailing lists.

Equipment required

In order to be able to install Pubcookie you will require the following:

1. A subscription to the Redhat network for at least one slot for Redhat AS3.0;
2. A standard "x86" server (i.e. a standard server with standard Intel or AMD chip set as sold by Dell etc.);
3. The ability to synchronise system time against a network time protocol (NTP) server;
4. A SSL server certificate for the web server, the certificate must be valid and be signed by Certificate Authority (CA) e.g. Thawte or Verisign;
5. Root user access to the server;
6. A preexisting windows 2003 active directory as a user name password store with either kerberos or LDAP access enabled.

Note

It is possible to install Pubcookie against any kerberised login, or against LDAP, or Unix passwd (`/etc/shadow`) and most of this guide would still be valid for these. However the guide will concentrate on Windows LDAP or kerberised login as that is likely to be the most common scenario encountered in UK Higher and Further Education establishments.

Assumptions

The following assumptions have been made:

- The Pubcookie login server is going to be a standalone web server that only serves Pubcookie login requests, i.e. it is not used for other secure web serving or for other tasks. As it will form the main gateway for web based logins it is imperative that it is secure as possible. The server should therefore have as few applications running as possible in order to reduce the number of potential exploits;

- The shell that you are using is bourne-shell compatible: if you are using a different shell (e.g. csh, tcsh, ksh) then you will need to substitute various commands for their equivalents (e.g. 'export' becomes 'setenv', etc.). The default shell for Redhat AS3.0 is bash, which is bourne-shell compatible.

Document Conventions

Code listings are provided within boxes like this:

```
$ echo hello world
```

File contents are provided within boxes like this:

```
The contents of a file
```

Some commands can be executed as a normal user, others are required to be executed with super user privileges. You can follow this guide entirely as the super-user, or you could use a normal user for a number of steps. For the latter case, switching between users and managing resulting permission issues is an exercise left to the reader.

To indicate which category a given command falls into, the standard bourne-shell delimiters are used, '$' for normal user, '#' for superuser:

```
$ echo normal user command
# echo super user command
```

Installation

Installation of Pubcookie relies on the installation of several packages and proper firewall configuration. While none of the steps are difficult in themselves, the combination of so many steps can be quite daunting. Hopefully the guide below will break down the installation process and ease the job of installing Pubcookie.

Prior Configuration

Remote File Systems (e.g. NFS)

Unless otherwise stated, do not build and install the components in this guide under a remote branch of the file system. All the examples here are installed under `/usr/local`. If `/usr/local` is part of a network-mounted or shared file system, replace this with something local, e.g. `/opt`.

This becomes important when having physically separate authentication and application servers, as described in 'Setting up an application server on a different box'.

Accessing the Web

Many of the following stages require obtaining files from the Internet. If necessary, configure your environment so that tools such as wget are able to access the web:

```
$ export http_proxy=http://webproxy.example.edu:8080
```

Where webproxy.example.edu is the host and 8080 the port of your web proxy.

Firewall settings

It is generally regarded as good practice to have a firewall installed on servers. Redhat AS3.0 comes with the "iptables" firewall solution. If you have your own firewalling solution or policy then you may wish to ignore this section, however, do note the ports that need to be unblocked.

It is probably a good idea to get the firewall setting sorted out as early as possible. It is the kind of thing you can forget later on in an install. Since firewalls drop connections silently it is difficult to identify them as the source of a problem.

In order to use Pubcookie you will need to allow:

1. Network time protocol, this usually occurs via UDP communication to port 123;
2. TCP communication with the Pubcookie keyserver, the default is to port 2222;
3. http and https web serving, the default ports are 80 and 443, communication is via TCP;
4. Kerberos client communication with a kerberos server UDP and TCP via port 88;

Note

It is possible to check what is listening on an open port using `lsof -i:<port number>`

There are lots of ways to setup firewalls and lots of situations which require different firewall settings. If you don't know how to setup a firewall in your institution, now is the time to consult someone who does. However if you are just experimenting the following rules would form the basis of a reasonable setup. Redhat keeps the iptables firewall rules in `/etc/sysconfig/iptables`. The syntax of this file is that generated by the tool `iptables-save`.

The following commands guarantee that the required ports are available:

```
# iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
# iptables -A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
# iptables -A INPUT -p udp -m udp --dport 123 -j ACCEPT
# iptables -A INPUT -p tcp -m tcp --dport 2222 -j ACCEPT
# iptables -A INPUT -p udp -m udp --dport 88 -j ACCEPT
# iptables -A INPUT -p tcp -m tcp --dport 88 -j ACCEPT
```

The following commands ensure that local and existing connections are uninhibited:

```
# iptables -A INPUT -m state --state RELATED -j ACCEPT
# iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
# iptables -A INPUT -i lo -m state --state NEW -j ACCEPT
```

The following commands ensure that any traffic not already dealt with by the above commands will be

dropped:

```
# iptables -P FORWARD DROP
# iptables -P INPUT   DROP
```

Caution

If you are remotely manipulating the machine, you may want to allow through the traffic that lets you do so. The secure shell protocol ssh operates on TCP port 22. Make sure you include a rule allowing this traffic before changing the default policy to 'DENY', especially if the machine is not physically close to you! The relevant line would be:

```
# iptables -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
```

Note

There is some excellent documentation on iptables and its parent project netfilter at http://www.netfilter.org/. Go there for help writing more complex iptable recipes.

NTP setup

Pubcookie requires accurate network time, it uses time stamps on cookies and forms to try and avoid "replay" attacks. In order to get accurate network time running it is a good idea to use xntpd to keep track of accurate time. Xntpd is a daemon that talks to a variety of time sources (as configured in ntp.conf) which can include specialised hardware clocks but in most case are just NTP servers which are in turn synchronised to accurate clocks. Running xntpd gives a much more stable and accurate time as it learns how inaccurate the local machine's clock is and can keep it reasonably well adjusted even if it loses contact with the remote servers.

Note

It may be possible to use a publicly accessible network time protocol server is you don't have access to one. The web site at http://www.eecis.udel.edu/~mills/ntp/clock2a.html may provide information about this. This has not been tested by the author so use it at your own risk.

To configure edit `/etc/ntp.conf` to tell it which NTP server to use. You can put in multiple NTP server entries for redundancy.

```
#
# Standard NTP configuration for  systems
#

server  ntpA.example.edu
server  ntpB.example.edu
server  ntpC.example.edu

#
# Drift file.  Put this in a directory which the daemon can write to.
# No symbolic links allowed, either, since the daemon updates the file
# by creating a temporary in the same directory and then rename()'ing
# it to the file.
#
driftfile /etc/ntp/drift
```

Then to get `xntpd` to run and survive server reboots you need to:

```
# chkconfig ntpd on
# service ntpd start
```

To check it is running properly run:

```
$ ntpq -p
```

You should see output something like:

```
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+ntp1.example.edu  ntp1.ja.net      2 u  605 1024  377    0.771   -0.796   0.170
 101.101.101.101   0.0.0.0         16 u    - 1024    0    0.000    0.000 4000.00
*ntp2.example.edu  ntp1.ja.net      2 u  298 1024  377    3.608   -2.041   1.324
 LOCAL(0)          LOCAL(0)        10 l   28   64  377    0.000    0.000   0.008
```

Apache Installation

In order to install Pubcookie you will require the httpd-devel version of apache httpd as supplied by Redhat. The "devel" version is required as the apxs script is needed for the Pubcookie install. Obviously openssl and mod_ssl are required to provide secure serving necessary for logins.

To install the required packages simply run up2date (as user root):

```
# up2date httpd
# up2date httpd-devel
# up2date openssl
# up2date openssl-devel
# up2date mod_ssl
```

This will get you Apache 2, Apache development files including the module installer (apxs), openssl for secure SSL and mod_ssl to enable SSL in Apache.

It is a good idea to de-activate Apache modules that are not being used to limit the amount of code that is being executed. The less code executed, the fewer potential security holes that are open to crackers.

If you are using the server for another purpose then you may need additional modules. De-activate other modules by editing `/etc/httpd/conf/httpd.conf`. We also suggest removing files (or renaming to remove the `.conf` extension) in `/etc/httpd/conf.d` which are unnecessary: e.g. `php.conf`, `perl.conf`,`python.conf`.

The following list of LoadModule directives will provide the required modules:

```
LoadModule access_module modules/mod_access.so
LoadModule auth_module modules/mod_auth.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule env_module modules/mod_env.so
LoadModule setenvif_module modules/mod_setenvif.so
LoadModule mime_module modules/mod_mime.so
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule dir_module modules/mod_dir.so
LoadModule alias_module modules/mod_alias.so
LoadModule cgi_module modules/mod_cgi.so
```

Note

The modules `mod_ssi.so` and `mod_pubcookie.so` are also required, but they are inserted into `/`

`etc/httpd/conf.d/ssl.conf` in the later stages of this guide.

It might be an idea to remove the following as well (unless you know you need it):

```
AddHandler imap-file map
AddOutputFilter INCLUDES .shtml
```

Start the Apache httpd server as a service (for details of service script see /etc/init.d/httpd):

```
# service httpd start
```

Now browse to the server via http (visit http://weblogin.example.edu/) and https (visit https://weblogin.example.edu/). In both cases you should see the "Redhat Enterprise Linux test page".

Optionally you can use run levels to make httpd fire up after reboot using chkconfig (for full details use "`man chkconfig`" and Redhat documentation). In this case the server runs at run level 2,3 so httpd will start once the server reaches those run levels. Those running graphical (non headless) systems may wish to set it so httpd also runs at the run level required by these systems.

```
# chkconfig --level 23 httpd on
# chkconfig --list
```

You may wish to decide on a log rotation and retention policy. The default Redhat install will result in log file that is rotated weekly and kept for a month.

Getting proper signed certificates

It is necessary to generate proper signed certificate and keypair in order to work with Pubcookie. This needs to be signed by a trusted Certificate Authority as, in the secure negotiation when generating a keypair, Pubcookie checks the signing chain to make sure that the requesting server is valid.

```
$ openssl genrsa -out shib.example.edu.key 1024
$ openssl req -new -key shib.example.edu.key -out shib.example.edu.csr
```

For generation of temporary self-signed certificate for testing, until you get the fully signed one back:

```
$ openssl x509 -req -days 30 -in shib.example.edu.csr -signkey shib.example.edu.ke
```

Note the is a Makefile at `/etc/httpd/conf` which will allow you to generate certificates and keypairs, however it is preferable to do it by hand in order to know the precise steps taken.

Choose a Authentication flavour

Pubcookie supports two "flavours" of authentication that can be used against Windows Active Directory. First and most easily you can use LDAP to validate username and password and this works well. Second you can use Kerberos which, while technically harder to achieve and support, can offer a more robust, secure and functional solution than LDAP.

LDAP Authentication flavour

LDAP is the easiest authentication flavour to install and support. Many institutions will already use it to communicate with their Active Directory. Pubcookie supports LDAP and secure LDAPS to query the

server. This guide will describe how to use LDAP, LDAPS appears not to be switched on by default in Windows 2003. Use of unencrypted LDAP means that the communication between the weblogin server and the Active Directory should be over a secure part of an institutions network as sniffers on compromised machines would be able to intercept unencrypted username and password communication between the klogin server and Active Directory. LDAPS would overcome this security problem.

To install LDAP clients for Pubcookie to work against run the following commands:

```
# up2date openldap
# up2date openldap-clients
# up2date openldap-devel
```

You then need to construct an ldap url for authenticating against your system. This will be used in the Pubcookie configuration file discussed later in this document. It is probably best to get the Active Directory administrator to construct this. It is generally in a form that will look something like this:

```
ldap_uri: ldap://activdir.example.edu:389/ou=All%20Users,dc=win,dc=example,dc=edu?
```

The ldap uri will be different for Active Directories in different institutions. The important thing to find out is the search base which ldap will look for user accounts (e.g.ou=All%20Users,dc=win,dc=example,dc=edu), whether it should search for users in sub branches (e.g. ?sub), and whether you have anonymous access or need to bind access with a username and password. If it uses anonymous access don't use the x-BindDN=bindId,x-Password=password bit. If you need to bind then get your active directory admin to setup an account to bind with. It is also important to find out what user accounts are called: in this example they are identified by "uid". Replace uid in the URL with whatever you define your accounts as e.g. If your accounts are identified by "cn" change "uid=%s" to "cn=%s".

LDAP URLs look like this:

```
ldap "://" [ host [ ":" port ] ] ["/" [ dn [ "?" [ attributeList ] [ "?" [ scope ]
```

Where:

- host and port specify where the LDAP server is located;
- dn is the LDAP Distinguished Name to search for;
- attributeList is a comma-delimited list of what to retrieve;
- scope is either base, one, or sub (base is the default if unspecified);
- filterString is the search filter;
- extensions is a comma-delimited list of extensions.

Note

The format of an LDAP URL is described in great detail in RFC 2255: http://www.faqs.org/rfcs/rfc2255.html

Note

URL-illegal and reserved characters, including spaces, commas and question marks must be escaped if occurring within the distinguished name and filter sections of the URL, or within an attribute. The escaping mechanism is described within section 2.2 of RFC 1738 (http://www.faqs.org/rfcs/rfc1738.html).

Briefly, spaces become '%20', commas '%2c' and question marks '%3f'.

Testing the LDAP URI

The program `ldapsearch` (from the openldap-clients package) can be used to ensure the LDAP URI that you have constructed is correct:

```
$ ldapsearch -h activedir.example.edu -x                  \
             -b "ou=All Users,dc=win,dc=example,dc=edu" \
             -D BindDN                                    \
             -w password                                  \
             '(cn=surname)'
```

Where:

- -h specifies the host of the LDAP server;
- -x specifies simple authentication (as appose to SASL;
- -b specifies the distinguished name to search for, prior to having URL-illegal and reserver characters substituted;
- -D specifies the username to bind with;
- -w specifies the password to bind with;
- the string '(cn=surname)' is the search filter (again, prior to URL character substitution).

Kerberos authentication flavour

Kerberos is a difficult authentication flavour to install and support. However it offers the advantage that it encrypts communication between the login server and the Active Directory; it will fall over if one of the Active Directory servers becomes unreachable; it supports proxying of Kerberos tickets. This latter feature may lead to a way of seamlessly authenticating access to portal "plugins", however this is relatively untried technology and will not be discussed further in this guide.

Kerberos is available at http://web.mit.edu/kerberos/www/. The version used in this guide is krb5-1.3.4, the download of which is available at http://web.mit.edu/kerberos/dist/krb5/1.3/krb5-1.3.4-i686-pc-linux-gnu.tar. In order for the Kerberos client to work properly with Windows Active Directory 2003, a version of Kerberos at least this recent needs to be used. The default Kerberos distribution that comes with Redhat AS3.0 is not recent enough. Problems will be encountered when large usernames/domains are encountered. The root of this problem is that when Windows sees a large username it decides it needs 2 network packets to send tickets relating to it. This causes it to switch from UDP communication to TCP, this confuses earlier Kerberos clients causing login failure. The more recent version, which has more robust UDP/TCP negotiation, can be installed in a non standard directory in order not to conflict with other kerberised services which may already be in use on the machine (e.g. kerberised login to user accounts).

```
$ mkdir /usr/local/kerb
$ cd /usr/local/kerb
$ wget http://web.mit.edu/kerberos/dist/krb5/1.3/krb5-1.3.4.tar
$ tar -xvf krb5-1.3.4.tar
$ rm krb5-1.3.4.tar
$ tar -xzvf krb5-1.3.4.tar.gz
$ cd krb5-1.3.4/src/
$ ./configure --prefix=/usr/local/kerb/krb5-1.3.4 --enable-shared --without-tcl
$ make
```

```
# make install
```

Note

on Redhat AS3.0, some TCL test utilities in the Kerberos package fail to build, so we disable them with the --without-tcl flag.

Note

Slightly unorthodox packaging is in use here: the tarball as downloaded contains a gzipped tarball of the same name, hence the rm step.
This recent version of Kerberos should be installed under `/usr/local/kerb/krb5-1.3.4`. It is now necessary to write a `/etc/krb5.conf` file. When writing this configuration file it is possible to use whichever enctypes you find suitable, however in our experience arcfour-hmac-md5 enabled us to use 8 bit characters (e.g. the £ sign) in passwords, as the default encryption rejected passwords using 8 bit characters. The `/etc/krb5.conf` file should look something like:

```
[logging]
        default = FILE:/var/log/kerberos/krb5lib.log

[libdefaults]
 ticket_lifetime = 600
 default_tkt_enctypes = arcfour-hmac-md5
 default_tgs_enctypes = arcfour-hmac-md5
 default_etypes = arcfour-hmac-md5
 default_etypes_des = arcfour-hmac-md5
 default_realm = REALM.EXAMPLE.EDU

[realms]
 CAMPUS.EXAMPLE.EDU = {
  kdc = ad1.example.edu:88
  kdc = ad2.example.edu:88
  kdc = ad3.example.edu:88
  kdc = ad4.example.edu:88
  default_domain = example.edu
 }

[domain_realm]
 .example.edu = REALM.EXAMPLE.EDU

[appdefaults]
 kinit = {
        renewable = true
        fowardable = true
        }
```

To get a Kerberos keytab file made for the domain you wish to authenticate against, see guide at http://www.microsoft.com/windows2000/techinfo/planning/security/kerbsteps.asp. Depending on your Active Directory configuration, the keytab may need to be created on the primary domain controller. Install the keytab at `/etc/krb5.keytab`.

It is possible to test the installation by using the compiled binaries:

```
$ cd /usr/local/kerb/krb5-1.3.4/bin
$ ./klist
$ ./kinit <username>
$ ./klist
$ ./kdestroy
```

Pubcookie installation

The Pubcookie installation process is described in the documents available at http://www.pubcookie.org/. If you wish to understand the theory of how Pubcookie works then refer to them. What follows here is a description of a working installation process specific to Redhat AS3.0 (most of the steps would probably work on other Linux distributions, however your mileage may vary). Depending on which authentication type you have chosen to use either proceed to the next step (install against ldap) or skip down the (install against Kerberos) section.

Installing Pubcookie from source against LDAP

This guide will install both the login server and the authentication module. In theory a live deployment would not have the authentication module installed on the same machine as the login program (under the theory that the login server should have as little running as possible to present the smallest "attack face" to crackers). However compiling and installing both allows for much easier initial testing. The authentication module can be deactivated at a later date if you feel it is a security risk.

```
$ mkdir /usr/local/pubcookie
$ cd /usr/local/pubcookie
$ wget http://www.pubcookie.org/downloads/pubcookie-3.1.1.tar.gz
$ tar -xvzf pubcookie-3.1.1.tar.gz
$ cd pubcookie-3.1.1/
$ ./configure --prefix=/usr/local/pubcookie/ \
              --enable-login                 \
              --enable-ldap                  \
              --with-apxs=/usr/sbin/apxs
$ make
# make install
```

Installing Pubcookie from source against Kerberos

This guide will install both the login server and the authentication module. As discussed for LDAP installation above this may cause the security paranoid some concern. You may wish to disable the application module when the system is deployed live. When compiling against Kerberos, for libraries that don't come in the default Redhat AS3.0 install it is necessary to force Pubcookie to use the new libraries, as it will try to use the pre-existing old Kerberos libraries. This can be done by setting the LDFLAGS and CPPFLAGS environment variables when compiling:

```
$ mkdir /usr/local/pubcookie
$ cd /usr/local/pubcookie
$ wget http://www.pubcookie.org/downloads/pubcookie-3.1.1.tar.gz
$ tar -xvzf pubcookie-3.1.1.tar.gz
$ cd pubcookie-3.1.1/

$ export KRBDIR=/usr/local/kerb/krb5-1.3.4
$ export CPPFLAGS=-I$KRBDIR/include
$ export LDFLAGS="-Xlinker -rpath $KRBDIR/lib -L$KRBDIR/lib"

$ ./configure --prefix=/usr/local/pubcookie/      \
              --enable-login                      \
              --enable-krb5                       \
              --with-krb5-dir=$KRBDIR             \
              --with-krb5-inc-dir=$KRBDIR/include \
              --with-krb5-lib-dir=$KRBDIR/lib     \
              --with-apxs=/usr/sbin/apxs
$ make
# make install
```

Note

In order to make this listing clearer and to save on possible typing errors, the environment variable KRBDIR is defined and used. This is not mandatory.
To confirm it has installed properly:

```
$ ls -lt /usr/local/pubcookie/login/index.cgi
```

The above command should list index.cgi as having been created at the time you ran make install. To confirm that index.cgi has compiled against the new Kerberos libraries and not used the default that comes with Redhat AS3.0 then use the command:

```
$ ldd /usr/local/pubcookie/login/index.cgi
        libssl.so.4 => /lib/libssl.so.4 (0xb75a9000)
        libcrypto.so.4 => /lib/libcrypto.so.4 (0xb74b2000)
        libdl.so.2 => /lib/libdl.so.2 (0xb74af000)
        libnsl.so.1 => /lib/libnsl.so.1 (0xb749a000)
        libkrb5.so.3 => /usr/local/kerb/krb5-1.3.4/lib/libkrb5.so.3 (0xb743b000)
        libc.so.6 => /lib/tls/libc.so.6 (0xb7303000)
        libcom_err.so.3 => /usr/local/kerb/krb5-1.3.4/lib/libcom_err.so.3 (0xb7301
        libgssapi_krb5.so.2 => /usr/local/kerb/krb5-1.3.4/lib/libgssapi_krb5.so.2
        libk5crypto.so.3 => /usr/local/kerb/krb5-1.3.4/lib/libk5crypto.so.3 (0xb72
        libresolv.so.2 => /lib/libresolv.so.2 (0xb72bc000)
        libz.so.1 => /usr/lib/libz.so.1 (0xb72ad000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb75eb000)
```

This should list index.cgi as linking against Kerberos at `/usr/local/kerb/kerb5-1.3.4` and not `/usr/kerberos`. In order to ensure that Apache runs index.cgi linking to the right libraries add the following to `/etc/httpd/conf.d/ssl.conf`

```
setEnv LD_LIBRARY_PATH /usr/local/kerb/krb5-1.3.4/lib
```

This directive should ensure that the LD_LIBRARY_PATH environment variable is passed to index.cgi when it is run, and this should force index.cgi to use the new Kerberos libraries.

You will also need to get your Active Directory administrator to generate a keytab for you. This is a file used by Kerberos to authenticate the server before it can start authenticating the user. Your active directory admin should know how to do this however helpful details can be found in the Microsoft knowledge base documentation at http://support.microsoft.com/default.aspx?scid=kb;en-us;Q324144. Be aware that the procedure may have changed for how to generate keytabs in Windows 2003. Your Active Directory administrator should be able to access information on this much more easily than we can.

Once a keytab is generated save it as /etc/krb5.keytab on your login server.

Continuing Installation

Now that you have generated an index.cgi that uses either ldap or kerberos to authenticate you need to install index.cgi in your cgi-bin, so that it can be reached by a web browser:

```
$ cp -p /usr/local/pubcookie/login/index.cgi /var/www/cgi-bin
```

and add the following to `/etc/httpd/conf.d/ssl.conf` if not already present to ensure the login cgi program works properly:

```
<Directory "/var/www/cgi-bin">
    SSLOptions +StdEnvVars
```

```
</Directory
```

Configuring Pubcookie

First copy the server certificate key pair and ca-bundle to Pubcookie dir (this prevents any problems with permissions on various directories leading to the key pair):

```
$ cp -p /etc/httpd/conf/ssl.crt/server.crt /usr/local/pubcookie/
$ cp -p /etc/httpd/conf/ssl.key/server.key /usr/local/pubcookie/
$ cp -p /etc/httpd/conf/ssl.crt/ca-bundle.crt /usr/local/pubcookie/
```

These files need to be set so that the apache user can read them:

```
chmod o+r /usr/local/pubcookie/shib.example.edu.key
chmod o+r /usr/local/pubcookie/shib.example.edu.crt
chmod o+r /usr/local/pubcookie/ca-bundle.crt
```

Pubcookie comes with config.login.sample, which is a sample config file, and config, which is the live config file. Changes should be made to the latter.

Below is a specimen config file. This config file will result in a high level of logging for Pubcookie error messages which can be useful when diagnosing problems, turn this down to "logging_level: 1" in a production service. This particular config will accept any username and password as valid, this is handy for testing purposes. When you wish to test real username/password verification comment out "basic_verifier: alwaystrue" and either enable the LDAP basic_verifier (commented out below) or the Kerberos basic_verifier (commented out below that):

```
# 1 is a good starting point
logging_level: 9

# for testing, an 'always true' verifier
basic_verifier: alwaystrue

## using the ldap verifier:
# basic_verifier: ldap
# ldap_uri: ldap://activdir.example.edu:389/ou=All%20Users,dc=win,dc=example,dc=ed

## using the kerberos verifier:
# basic_verifier: kerberos_v5
# kerberos5_keytab:/etc/krb5.keytab

# granting keypair
granting_key_file: /usr/local/pubcookie/keys/pubcookie_granting.key
granting_cert_file: /usr/local/pubcookie/keys/pubcookie_granting.cert

# login server config
login_uri: https://weblogin.example.edu/cgi-bin/index.cgi
login_host: weblogin.example.edu
enterprise_domain: .example.edu
logout_prog: /logout/index.cgi

# keyserver config
keymgt_uri: https://weblogin.example.edu:2222
keyserver_client_list: weblogin.example.edu appserver1.example.edu appserver2.exam
ssl_key_file: /usr/local/pubcookie/server.key
ssl_cert_file: /usr/local/pubcookie/server.crt
ssl_ca_file: /usr/local/pubcookie/ca-bundle.crt
```

```
# site-specific policies
PubcookieInactiveExpire: 300
default_l_expire: 8h

# custom logout msgs
app_logout_string-appserver.example.edu-testapp: <font size="+1">Testapp logout wo
app_logout_string-webmail.example.edu-webmail: <font size="+1">Webmail Logout Succ
```

Testing the weblogin server

You should now have a functioning weblogin server. In order to test this you can try to login in using the weblogin (remember to start the httpd server: "service httpd start" would probably do it). This test is called a "pinit" test and is named after the Kerberos kinit.

Please note that the sample login pages contain broken images and links to non-existent URLs. The precise appearance of these pages (and the standard Apache error pages) should be fine-tuned for your institution. This is covered in section 2.6.7 'Setting up the login pages and error pages'.

With a browser (Internet Explorer, Firefox, etc.), browse to the location of your login page e.g. https://weblogin.example.edu/cgi-bin/index.cgi and login. You should be able to login. Look through the logs at `/var/log/secure` to see how it progressed. This test is only suitable for checking that the install worked. In order to get application servers to use the login server it is necessary to setup the keyserver that comes with Pubcookie to handle the generation and distribution of keys.

Setting up the keyserver

To add the keyserver to your system create a file called `/etc/xinetd.d/keyserver`. Copy the following into the file:

```
service keyserver
{
type = UNLISTED
protocol = tcp
port = 2222
disable = no
socket_type = stream
wait = no
user = root
group = tty
server = /usr/local/pubcookie/keyserver
}
```

Then restart `Xinetd` so that the changes take effect:

```
# service xinetd restart
Stopping xinetd:                                           [  OK  ]
Starting xinetd:                                           [  OK  ]
```

It is possible to check that the service is listening by listing open files on the port the keyserver listens on (port 2222):

```
# lsof -i:2222
COMMAND   PID USER   FD   TYPE DEVICE SIZE NODE NAME
xinetd  24961 root    6u  IPv4 169627       TCP *:2222 (LISTEN)
```

If you don't get a listing it is likely that the keyserver hasn't started and you should check the xinetd configuration (in particular the contents of file `/etc/xinetd.d/keyserver`).

It may also be necessary to add entries into `/etc/hosts.allow` to allow Pubcookie application servers to make requests to the keyserver. Your host.allow configuration will probably allow connections but if connections aren't working this is a good place to check:

```
keyserver: weblogin.example.edu, appserver2.example.edu, appserver1.example.edu
```

In order for the login server to be able to encrypt and sign cookies it needs its own granting certificate and keypair. This keypair doesn't require signing by a Certificate Authority so it can be created quite simply with the following commands:

```
$ cd /usr/local/pubcookie/keys
$ openssl req -new -x509 -out pubcookie_granting.cert \
  -newkey rsa:1024 -nodes -keyout pubcookie_granting.key
Generating a 1024 bit RSA private key
..........++++++
.....................................................................++++++
writing new private key to 'pubcookie_granting.key'
-----
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) [GB]:GB
State or Province Name (full name) [Berkshire]:Some province
Locality Name (eg, city) [Newbury]:Some city
Organization Name (eg, company) [My Company Ltd]:University of Somewhere
Organizational Unit Name (eg, section) []:Computing service
Common Name (eg, your name or your server's hostname) []:weblogin.example.edu
Email Address []:webmaster@example.edu
```

Testing the keyserver and application server

In order to check that the keyserver is functioning properly it is a good idea to test the keyserver with a request from the login server (this is easier to debug than requesting from another box). Note this will only work if you built the Pubcookie module as well as the login program (if you have followed this guide, then you will have built both). To generate a key that Pubcookie will use to encrypt traffic between it and this application server run the following commands:

```
$ cd /usr/local/pubcookie
$ cp -p starter.key keys/weblogin.example.edu
$ ./keyclient
make_crypt_keyfile: hello
make_crypt_keyfile: goodbye
Set crypt key for weblogin.example.edu
```

The file `/usr/local/pubcookie/keys/weblogin.example.edu` should now contain undecipherable text instead of the repeated word keys (the text it had before keyclient). If this process appears to have failed look for errors in the files `/var/log/messages` and `/var/log/secure`.

To test this create a directory called `/var/www/html/test`, create an index.html file in it and write some text in that file (e.g. "helloWorld!") then add the following to your `/`

etc/httpd/conf.d/ssl.conf:

```
<Directory /var/www/html/test >
AuthType EGNetID
require valid-user
</Directory>
```

This directive should require that all visitors to anything existing in the test directory should login first.

It is also necessary to alter the Apache config file to load the Pubcookie module for authentication. Firstly the module must be loaded into the Load module section of /etc/httpd/conf.d/ssl.conf:

```
LoadModule ssl_module modules/mod_ssl.so
LoadModule pubcookie_module modules/mod_pubcookie.so
```

Then add the configuration directives after the LoadModule directive:

```
<IfModule mod_pubcookie.c>

PubcookieGrantingCertFile /usr/local/pubcookie/keys/pubcookie_granting.cert
PubcookieSessionKeyFile /etc/httpd/conf/ssl.key/server.key
PubcookieSessionCertFile /etc/httpd/conf/ssl.crt/server.crt
PubcookieLogin https://weblogin.example.edu/cgi-bin/index.cgi
PubcookieDomain .example.edu
PubcookieKeyDir /usr/local/pubcookie/keys/
PubcookieAuthTypeNames EGNetID

## Disable inactivity timeout by default to not timeout
<Directory "/var/www/html">
PubcookieInactiveExpire -1
</Directory>NCL.AC.UK

</IfModule>
```

Note

EGNetID is an authentication type name that is defined by us and the two occurances (one within this If-Module block and one within the Directory block above) could be replaced with any other string. However, the official Pubcookie documentation uses EGNetID, so we are too for the sake of consistency.

Restart the Apache server so it takes into account the configuration change ("service httpd restart" should do it). Browse to the index page you just created e.g. https://weblogin.example.edu/test/index.html. If it has worked you should be redirected to the login page https://weblogin.example.edu/cgi-bin/index.cgi where you will login. If successful you will be redirected back to https://weblogin.example.edu/test/index.html and should be able to see "helloWorld!". If unsuccessful look in the logs in the files /var/log/secure, /var/log/messages and in the directory /var/log/httpd for possible explanations.

Setting up the login pages and error pages

While you now have a fully working server, the login pages will be the default, which are specific to University of Washington. Users will also be subject to the default Apache 405 error pages if they try to access a Pubcookie protected resource which they are not allowed to access. It is a good idea to edit the file at /usr/local/pubcookie/login_templates and to setup a useful set of Apache error

pages (see Apache documentation, section 'Custom Error Responses': http://httpd.apache.org/docs-2.0/custom-error.html).

Setting up an application server on a different box

Setting up an application server on a different box is relatively straightforward. Follow these steps, as before, prior to continuing:

- Configure the firewall as in 'Firewall Settings', with the exception of allowing Kerberos traffic (unless you need Kerberos traffic for another purpose e.g. obtaining a command prompt on the machine)
- Setup NTP as in the 'NTP Setup' section
- Install Apache as in the 'Apache Installation' section

Pubcookie installation is relatively straightforward:

```
$ mkdir /usr/local/pubcookie
$ cd /usr/local/pubcookie
$ wget http://www.pubcookie.org/downloads/pubcookie-3.1.1.tar.gz
$ tar -zxvf pubcookie-3.1.1.tar.gz
$ cd pubcookie-3.1.1
$ ./configure --prefix=/usr/local/pubcookie \
              --enable-apache               \
              --with-apxs=/usr/sbin/apxs
$ make
# make install
```

A specimen config file for Pubcookie is shown below:

```
# 1 is a good starting point
logging_level: 9

# SSL session keypair
ssl_key_file: /usr/local/pubcookie/server.key
ssl_cert_file: /usr/local/pubcookie/server.crt
ssl_ca_file: /usr/local/pubcookie/ca-bundle.crt
granting_cert_file: /usr/local/pubcookie/keys/pubcookie_granting.cert

# login server config
login_uri: https://weblogin.example.edu/cgi-bin/index.cgi
login_host: weblogin.example.edu
enterprise_domain: .example.edu
logout_prog: /logout/index.cgi

# keyserver config
keymgt_uri: https://weblogin.example.edu:2222
keyserver_client_list: weblogin.example.edu appserver1.example.edu appserver2.exam

# site-specific policies
PubcookieInactiveExpire: 300
default_l_expire: 8h

# custom logout msgs
app_logout_string-appserver.example.edu-testapp: <font size="+1">Testapp logout wo
app_logout_string-webmail.example.edu-webmail: <font size="+1">Webmail Logout Succ
```

Note

The SSL key and certificate files listed here are the ones belonging to the Application Server, and must not be confused with those of the Authentication Server.

Make changes to the apache config as described in the section 'Testing the keyserver and application server' above. This should result in a working Pubcookie application server. The only difference is that you need to transfer the pubcookie_granting.cert file to the application server (default location is `/usr/local/pubcookie/keys/pubcookie_granting.cert`) via SFTP or whatever mechanism you are comfortable with. Then tell the keyserver on the login server to accept connections from this application server .

On the login server:

```
$ cd /usr/local/pubcookie/
$ ./keyclient -P appserver1.example.edu
```

It may also be necessary to edit the hosts.allow file at /etc/hosts.allow to allow communication between the keyserver and the application server by adding something like:

```
keyserver: appserver1.example.edu
```

Then request a key from the application server:

```
$ cd /usr/local/pubcookie
$ ./keyclient
make_crypt_keyfile: hello
make_crypt_keyfile: goodbye
Set crypt key for appserver1.example.edu
```

If this fails look through the logs in `/var/log/secure` and `/var/log/messages` on both machines to get error messages. If successful this should result in a fully working application server.

Note

It should be possible to set up the appserver and login server to distribute the pubcookie_granting cert automatically however attempts to do this failed. Similarly it should not be necessary to use the permit (-P) option on the login server as the keyserver_client_list should inform it which server can request keys. However in practice this step proved to be necessary.

Warning

As described above in 'Remote File Systems (e.g. NFS)', Pubcookie directories should not be mounted on NFS. This is due to the security model built into the Pubcookie design: if a cracker takes control of an application server, they haven't automatically compromised the other application servers, or the authentication server. This is because each server shares its own enc. However if they had a common set of keys and certificates, all of the sharing servers would be un-trustworthy.

Redundant load balanced setup with easy disaster recovery.

It is possible to have two login servers load balanced by DNS round robin or other mechanisms. This may be desirable in that it removes the single point of failure aspect of having one login server for your single sign on infrastructure. Details on how to achieve this will be updated here once the technique is stable (it is being tested at present in the authoring institution).