Surviving Cyrus SASL

©2007 Patrick Koetter & Ralf Hildebrandt

state-of-mind

LISA’07
Dallas, November 2007

1 The Goal
2 Architecture
   ■ Components
   ■ Protocols
   ■ Areas of Authentication
3 Cyrus SASL
   ■ What is Cyrus SASL?
   ■ How Cyrus SASL works
      ■ libsasl in Client-Application
      ■ libsasl in Server-Application
   ■ Configuration
   ■ Testing
4 Practice
   ■ saslauthd authentication using shadow passwords
   ■ sasldb authentication
   ■ sql authentication
   ■ ldapdb authentication
Surviving Cyrus SASL

1. The Goal
2. Architecture
   - Components
   - Protocols
   - Areas of Authentication
3. Cyrus SASL
   - What is Cyrus SASL?
   - How Cyrus SASL works
     - libsasl in Client-Application
     - libsasl in Server-Application
   - Configuration
   - Testing
4. Practice
   - saslauthd authentication using shadow passwords
   - sasldb authentication
   - sql authentication
   - ldapdb authentication
Surviving Cyrus SASL

1 The Goal
2 Architecture
   - Components
   - Protocols
   - Areas of Authentication
3 Cyrus SASL
   - What is Cyrus SASL?
   - How Cyrus SASL works
     - libSASL in Client-Application
     - libSASL in Server-Application
   - Configuration
   - Testing
4 Practice
   - saslauthd authentication using shadow passwords
   - saslauthd authentication
   - sql authentication
   - ldapdb authentication
Cyrus SASL is
- an authentication-framework
- an implementation of SASL, the “Simple Authentication and Security Layer”
- standardised
- described in RFC 2222
- “the child of those sitting on the standard”

Application Range

- Cyrus SASL does not act on its own.
- Embedded into an connection-oriented application (e.g. SMTP, FTP, POP3, IMAP, LDAP)
- Cyrus SASL provides a protocol, which

  (...) includes a command for identifying and authenticating a user to a server and for optionally negotiating protection of subsequent protocol interactions. If its use is negotiated, a security layer is inserted between the protocol and the connection.
Advantages

Integrating Cyrus SASL in an application:

- simplifies software development
- provides stable and reliable functionality
- increases interoperability with other RFC compliant software

Disadvantages

Using Cyrus SASL in an application:

- drives users nuts, because the existing documentation focuses on developers
- may not get you far, because many things are undocumented
- is hard to memorize, because everything is handled differently
Cyrus SASL provides the `libsasl` library to developers

Developers link the library into their application

Mode (client- or server-mode) determines what `libsasl` will do for the application

**libsasl in Client-Application**

Tasks

- determine which mechanism the client must use during authentication
- process the tasks required by the mechanism
Tasks

- identify a list of mechanisms the server may offer
- process the tasks required by a chosen mechanism
- hand over authentication data to a password verification service
- notify server of authentication result

SASL-Terms used in authentication

- Client and server use an authentication interface to communicate
- They use mechanisms to exchange authentication data
- A password verification service or a method verify data in an authentication backend
- The server sends the authentication result to the client
- The server may authorize the client to take some action
Authentication Interface

The place where client and server meet to exchange authentication data and information:

- The application protocol defines client-server communication
- SASL is a framework for many applications. It must be free from application-specific protocol requirements
- Application protocols must specify client- and server-commands to carry out authentication
- libsasl is the glue for application-specific commands and universal SASL-routines

Mechanisms

Mechanisms define strategies for sending authentication data.

*SASL mechanism names must be registered with the IANA*

http://www.iana.org/assignments/sasl-mechanisms
Grouping by Characteristics

Mechanisms are grouped by similar characteristics:
- **Processing**
  - How is authentication processed?
- **Data**
  - Which data are send during authentication?
- **Security**
  - Which level of security can be achieved from the various processing <-> data combinations?

Groups of Mechanisms

Similar characteristics serve to group mechanisms:
- Anonymous-mechanism
- Plaintext-mechanisms
- Shared-Secret-mechanisms
- Ticket-mechanisms
- External-mechanisms
Anonymous-mechanism

- **Procedure**
  Client sends a string.

- **Data**
  Mechanism expects 'some' data. Think anonymous FTP!

- **Security**
  Anyone being capable to send some data is considered authenticated

- **Available mechanisms**
  - ANONYMOUS

Plaintext-mechanisms

- **Procedure**
  Mechanism encodes authentication data base64 (some transports are not 8-bit clean).

- **Data**
  Plaintext-mechanisms send authentication ID, authorization ID, password and (maybe) realm.

- **Security**
  - Transport is unencrypted (may be encrypted using TLS)
  - Authentication data must be stored on the server.

- **Available mechanisms**
  - PLAIN
  - LOGIN
authcid, authzid and password will be base64-encoded and sent as one string

```perl
# perl -MMIME::Base64 -e 'print encode_base64("username\0username\0password");'
dXNlcm5hbWUAdXNlcm5hbWUAcGFzc3dvcmQ=
```

**Example (SMTP)**

```plaintext
220 mail.example.com ESMTP Postfix
EHLO example.com
250-mail.example.com
250-PIPELINING
250-SIZE 10240000
250-AUTH DIGEST-MD5 CRAM-MD5 GSSAPI PLAIN LOGIN
250-AUTH=DIGEST-MD5 CRAM-MD5 GSSAPI PLAIN LOGIN
250-XVERP
250 8BITMIME
AUTH PLAIN dXNlcm5hbWUAdXNlcm5hbWUAcGFzc3dvcmQ=
235 Authentication successful
QUIT
221 Bye
```
LOGIN

- Username, password and optionally the domainname will be base64-encoded separately and also sent separately.
- LOGIN is a proprietary Microsoft mechanism. It is not standardised and documentation is not freely available.
- Outlook and Outlook Express can’t do PLAIN, but they can do LOGIN.

Example (SMTP)

```
220 smtp.example.com ESMTP server ready
EHLO test.example.com
250-smtp.example.com
250-STARTTLS
250 AUTH LOGIN CRAM-MD5
AUTH LOGIN
334 VXNlciBOYW1lAA== # User Name
dGlt # Tim
334 UGFzc3dvcmQA # Password
dGFuc3RhYWZ0YW5zdGFhZg== # tanstaaftanstaaf
235 Authentication successful.
```
Shared-Secret-mechanisms

- **Procedure**
  - Shared-Secret-mechanisms are Challenge-Response methods:
    - The server produces a challenge. The client can only solve (response) it, if it uses identical authentication data.

- **Data**
  - Username and challenge are encrypted using the password.
  - The complete string will be sent base64-encoded.
  - The password is never sent.

Shared-Secret-mechanisms II

- **Security**
  - Data is transported encoded and encrypted
  - Authentication data must be stored on the server
  - The password must be stored in plaintext format

- **Available Mechanisms**
  - CRAM-MD5
  - DIGEST-MD5
  - NTLM
EXTERNAL relies on external mechanisms that are not part of SASL

The server uses information, external to SASL, to determine whether the client is authorized to authenticate as the authorization identity. If the client is so authorized, the server indicates successful completion of the authentication exchange; otherwise the server indicates failure.

TLS

TLS is the only EXTERNAL-mechanism met “in the wild”.

- TLS offers client- and server-authentication using certificates.
- TLS encrypts the transport layer.
Ticket-mechanisms

- **Procedure**
  - Client authenticates with Kerberos-server and receives a ticket granting ticket.
  - The ticket granting ticket enables the client to request a ticket that grants usage of a service.

- **Data**
  - Client sends username and password to Kerberos-server.
  - Client sends only ticket granting ticket to gain access to service.

Ticket-mechanisms II

- **Security**
  - Neither username nor password are sent during SASL authentication.

- **Available Mechanisms**
  - Kerberos_4 (vulnerable, don’t use it)
  - GSSAPI (Kerberos_5) “the” secure mechanism
Password Verification Service

Password Verification Services verify authentication data on behalf of libsasl.

Advantages

- run as standalone daemons on the server
- may be run with special privileges (while the server application uses least privileges)
- may access authentication backends requiring special privileges

Disadvantages

- can only handle “insecure” plaintext-mechanisms

Available Password Verification Services

- pwcheck
- saslauthd
pwcheck

- pwcheck is the old, original Cyrus SASL Password Verification Service
- was used until end of Cyrus SASL 1.5.xx series
- is still part of the Cyrus SASL source tree
- pwcheck is deprecated

saslauthd

saslauthd is the official, current Cyrus SASL Password Verification Service. It can access various authentication backends:

```
# saslauthd -v
saslauthd 2.1.22
authentication mechanisms: \
  getpwent kerberos5 pam rimap shadow ldap
```
saslauthd authentication backends

- getpwent
  Access passwd

- kerberos5
  Authenticate against local Kerberos realm

- pam
  Send request to Pluggable Authentication Modules (PAM) and use result

- rmap
  Attempt login to remote IMAP-server.

- shadow
  Access shadow-file.

- ldap
  Authenticate (simple bind) with LDAP-server

Auxiliary Property Plugins

verify authentication data on behalf of libsasl (and they may do more...)

Advantages

- may access a variety of authentication backends
- may also write (create, modify) to authentication backends
- can do proxy authentication
- may use all available mechanism groups

Disadvantages

- run as user of the server that calls libsasl.
  Is that really a disadvantage?
Available Auxiliary Property Plugins

- sasldb
- sql
- ldapdb

sasldb

sasldb is the Cyrus SASL default authentication backend:

- sasldb is a Berkeley DB
- sasldb database format was changed from Cyrus SASL version 1.x to 2.x in order to make offering Shared-Secret mechanisms possible
  Use Berkeley DB tools to migrate
- since Cyrus SASL 2.x passwords are stored in sasldb as plaintext.
sasldb Utilities

- saslpasswd2
  - Create sasldb2
  - Create accounts in sasldb2
  - Modify accounts in sasldb2

- sasldblistusers2
  - List sasldb2-users

sql

sql is a generic driver to access various SQL-servers:
- MySQL
- PostgreSQL
- SQLite
Typical Problems

Accessing the SQL-server via PAM, in order to store passwords in encrypted form.
The same people don’t seem to mind sending username and password unencrypted over the wire...
The “frost”-patch “fixes” unencrypted storage in the SQL-server at the price of losing shared-secret mechanisms.
http://frost.ath.cx/software/cyrus-sasl-patches/

Idapdb

Idapdb is a driver to access the OpenLDAP server.
The driver implements proxy authentication as described in RFC 2222:

The separation of the authorization identity from the identity in the client’s credentials. This permits agents such as proxy servers to authenticate using their own credentials, yet request the access privileges of the identity for which they are proxying.
Idapdb II

Idapdb requires configuring Cyrus SASL authentication twice:
- Login of Cyrus SASL Idapdb-Plugin to slapd
- Usage of Idapdb-Plugin within server application
What needs to be configured?

- **Client**
  - only needs the credentials
  - The client (not SASL) may want to avoid certain mechanisms

- **Server**
  Server applications must be configured before Cyrus SASL serves them. A server application sends two values to `libsasl`:
  - `application_name` specifies part of the string used to identify the server-specific configuration file
  - `service_name` specifies the service (protocol) `libsasl` serves. PAM uses the service name to identify the service-specific configuration file.

Parameters

Cyrus SASL knows generic and method-specific parameters. Parameters that are specific to a method must be specified:

- on the command line when a password verification service is used
- in an `application_name.conf` when `auxprop-plugins` are used
## Generic Parameters

- **log_level**
  
  Controls the level of verbosity of messages sent to the syslogd service.

<table>
<thead>
<tr>
<th>Level</th>
<th>Verbosity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no messages</td>
</tr>
<tr>
<td>1</td>
<td>unusual errors</td>
</tr>
<tr>
<td>2</td>
<td>all authentication errors</td>
</tr>
<tr>
<td>3</td>
<td>log non-fatal warnings</td>
</tr>
<tr>
<td>4</td>
<td>more verbose than 3</td>
</tr>
<tr>
<td>5</td>
<td>more verbose than 4</td>
</tr>
<tr>
<td>6</td>
<td>traces of internal protocols</td>
</tr>
<tr>
<td>7</td>
<td>traces of internal protocols, including passwords</td>
</tr>
</tbody>
</table>

## Generic parameters II

Logging is inconsistent

No password verification service or auxprop-plugin implements all log levels. Some don’t log at all...
Generic parameters III

- **pwcheck_method**
  Specifies one or more password verification services and/or auxprop-plugins to process authentication. Valid values are the names of the password verification services or auxprop-plugins.

- **mech_list**
  Specifies a list of mechanisms a Cyrus SASL may offer a server. Valid values are the names of mechanisms, separated by whitespace.

Method-specific Parameters

... hold on. We’ll take a look at them when practice...
Tools for testing

- Testing Cyrus SASL isolated is important! Without you’ll have a hard time to tell if the error is in Cyrus SASL or the server that offers authentication.
- Many admins spend days looking for the error in their application...
- Problem
  Cyrus SASL has no “tools” to test!
testsaslauthd

**Problem**
Successful testing does not prove that all parts of the Cyrus SASL framework are working okay, because testsaslauthd does not (!) use the Cyrus SASL mechanism libraries...

**Command**

```
# testsaslauthd
```
```
testsaslauthd: usage: testsaslauthd -u username -p password
          [-r realm] [-s servicename]
          [-f socket path] [-R repeatnum]
```
Surviving Cyrus SASL

Server

```
# ./sample-server -h

- b ... #bits to use for encryption
  min=N minimum #bits to use (1 => integrity)
  max=N maximum #bits to use
- e ... assume external encryption
  ssf=N external mech provides N bits of encryption
  id=ID external mech provides authentication id ID
- m MECH force use of MECH for security
- f ... set security flags
  noplain require security vs. passive attacks
  noactive require security vs. active attacks
  nodict require security vs. passive dictionary attacks
  forwardsec require forward secrecy
  maximum require all security flags
  passcred attempt to receive client credentials
- i ... set IP addresses (required by some mechs)
  local=IP;PORT set local address to IP, port PORT
  remote=IP;PORT set remote address to IP, port PORT
- p PATH colon-separated search path for mechanisms
- s NAME service name to pass to mechanisms
- d DOM local server domain
- u DOM user domain
- l enable server-send-last
```

Client

```
# ./sample-client -h

- b ... #bits to use for encryption
  min=N minimum #bits to use (1 => integrity)
  max=N maximum #bits to use
- e ... assume external encryption
  ssf=N external mech provides N bits of encryption
  id=ID external mech provides authentication id ID
- m MECH force use of MECH for security
- f ... set security flags
  noplain require security vs. passive attacks
  noactive require security vs. active attacks
  nodict require security vs. passive dictionary attacks
  forwardsec require forward secrecy
  maximum require all security flags
  passcred attempt to pass client credentials
- i ... set IP addresses (required by some mechs)
  local=IP;PORT set local address to IP, port PORT
  remote=IP;PORT set remote address to IP, port PORT
- p PATH colon-separated search path for mechanisms
- r REALM realm to use
- s NAME service name to pass to mechanisms
- n FQDN server fully-qualified domain name
- u ID user (authorization) id to request
- a ID id to authenticate as
- d Disable client-send-first
- l Enable server-send-last
```
Surviving Cyrus SASL

Practice

1 The Goal
2 Architecture
   - Components
   - Protocols
   - Areas of Authentication
3 Cyrus SASL
   - What is Cyrus SASL?
   - How Cyrus SASL works
     - libsasl in Client-Application
     - libsasl in Server-Application
   - Configuration
   - Testing
4 Practice
   - saslauthd authentication using shadow passwords
   - sasldb authentication
   - sql authentication
   - ldapdb authentication

Procedure:
   - Prepare saslauthd environment
   - Create user test
   - Test
     - using testsaslauthd
     - using sample-server und sample-client
   - configure AUTH
   - in Postfix
   - in Cyrus IMAP
Saslauthd

# /usr/sbin/saslauthd -h
usage: saslauthd [options]
option information:
  -a <authmech>  Selects the authentication mechanism to use.
  -c            Enable credential caching.
  -d            Debugging (don’t detach from tty, implies -V)
  -r            Combine the realm with the login before passing to
                authentication mechanism. Ex. login: "foo" realm: "bar"
                will get passed as login: "foo@bar". The realm name is
                passed untouched.
  -O <option>   Optional argument to pass to the authentication
                mechanism.
  -l            Disable accept() locking. Increases performance, but
                may not be compatible with some operating systems.
  -m <path>     Alternate path for the saslauthd working directory,
                must be absolute.
  -n <procs>    Number of worker processes to create.
  -s <kilobytes> Size of the credential cache (in kilobytes)
  -t <seconds>  Timeout for items in the credential cache (in seconds)
  -V            Display version information and available mechs
  -V            Display verbose logging
  -h            Display this message.

Preparing Saslauthd environment

A classic...
The socket directory (run_path) is missing...

# /usr/sbin/saslauthd -d -a shadow
saslauthd[20983] :main : num_procs : 5
saslauthd[20983] :main : mech_option: NULL
saslauthd[20983] :main : run_path : /var/run/saslauthd
saslauthd[20983] :main : auth_mech : shadow
saslauthd[20983] :main : could not chdir to: /var/run/saslauthd
saslauthd[20983] :main : chdir: No such file or directory
saslauthd[20983] :main : Check to make sure the directory exists and is
saslauthd[20983] :main : writeable by the user this process runs as.
Create user test:

# useradd test
# passwd test

Test using testsaslauthd:

# testsaslauthd -u test -p -test -s smtp

Testing using sample-server and sample-client – sample-server sends sample as application_name:
/usr/lib/sasl2/sample.conf

pwcheck_method: saslauthd
mech_list: PLAIN LOGIN

Testing II

Start both applications in different terminals:
Terminal 1

# sample-server -p 8000 -s rcmd -m PLAIN

Terminal 2

# sample-client -p 8000 -s rcmd -m PLAIN localhost
There are two ways application specific configuration options can be given to Cyrus SASL:

- store them in a separate configuration file located in
  /usr/lib/sasl2.
  Since 2.1.22 --with-configdir configure option made
  the location configurable and defaults to /etc/sasl2.
- let server read configuration options from its own
  configuration file and pass them on when it calls libsasl.

Postfix uses a separate configuration file. By default it sends
the (configurable) application_name smtpd to libsasl:
/usr/lib/sasl2/smtpd.conf

pwcheck_method: saslauthd
mech_list: PLAIN LOGIN
Cyrus IMAP passes options to `libsasl` from its own configuration file `/etc/imapd.conf`:

```
sasl_pwcheck_method: saslauthd
sasl_mech_list: PLAIN LOGIN
```

Procedure:
- Create `sasldb2`
- Test using `sample-server` and `sample-client`
- configure AUTH
- in Postfix
- in Cyrus IMAP
# saslpasswd2 -h
This product includes software developed by Computing Services at Carnegie Mellon University (http://www.cmu.edu/computing/).
saslpasswd2: usage: saslpasswd2 [-v] [-c [-p] [-n]] [-d] [-a appname]
-p pipe mode -- no prompt, password read on stdin
-c create -- ask mechs to create the account
-d disable -- ask mechs to disable/delete the account
-n no userPassword -- don’t set plaintext userPassword property (only set mechanism-specific secrets)
-f saslDb use given file as sasldb
-a appname use appname as application name
-u DOM use DOM for user domain
-v print version numbers and exit

Creating saslDb

# saslpasswd2 -c -u example.com test
Password:
Again (for verification):
Listing saslbd content

```bash
# saslbdlistusers2 -h
This product includes software developed by Computing Services at Carnegie Mellon University (http://www.cmu.edu/computing/).
    -f saslbd    use given file as saslbd
    -v           print version numbers and exit
# saslbdlistusers2
test@example.com: userPassword
```

Testing

```
sample-server sends sample as application_name.
/usr/lib/sasl2/sample.conf:
pwcheck_method: auxprop
auxprop_plugin: saslbd
mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5
```
Testing II

Start both applications from separate terminals:

Terminal 1

    # sample-server -p 8000 -s rcmd -m PLAIN

Terminal 2

    # sample-client -p 8000 -s rcmd -m PLAIN localhost

Question

Do more secure mechanisms work?
Surviving Cyrus SASL

Practice

sasldb authentication

Configuring AUTH

**Postfix** (/usr/lib/sasl2/smtpd.conf):

- pwcheck_method: auxprop
- auxprop_plugin: sasldb
- mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5

**Cyrus IMAP** (/etc/imapd.conf):

- sasl_pwcheck_method: auxprop
- sasl_auxprop_plugin: sasldb
- sasl_mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5

Procedure:

- Create SQL database
- Create SASL sql configuration
- Test using sample-server and sample-client
- configure AUTH
- in Postfix
- in Cyrus IMAP
Creating the SQL database

```sql
mysql> describe user;
+----------+------------+------+-----+-------------+-------+
| Field    | Type       | Null | Key | Default | Extra |
|----------+------------+------+-----+-------------+-------+
| user     | char(255)  | YES  | UNI | NULL      |       |
| realm    | char(255)  | YES  |     | example.com|       |
| password | char(50)   | NO   |     | Mnd0dO5x  |       |
| active   | tinyint(1) | YES  |     | 1         |       |
+----------+------------+------+-----+-------------+-------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM user;
+------+-------------+----------+--------+
| user | realm        | password | active |
|------|--------------|----------|--------+
| test | example.com  | Mnd0dO5x | 1      |
+------+-------------+----------+--------+
1 row in set (0.00 sec)
```

Postfix SASL sql configuration

Postfix (/usr/lib/sasl2/smtpd.conf):

```conf
pwcheck_method: auxprop
auxprop_plugin: sql
mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5

# SQL configuration
sql_engine: mysql
sql_hostnames: localhost
sql_database: sasl
sql_user: sasl
sql_passwd: X6jx5nZe
sql_select: SELECT userpassword FROM user WHERE 
            username = '%u' AND 
            userrealm = '%r' AND 
            auth = '1'
sql_usessl: no
```
Cyrus IMAP SASL sql configuration

Cyrus IMAP (/etc/imapd.conf):

```
sasl_pwcheck_method: auxprop
sasl_auxprop_plugin: sql
sasl_mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5

# SQL configuration
sasl_sql_engine: mysql
sasl_sql_hostnames: localhost
sasl_sql_database: sasl
sasl_sql_user: sasl
sasl_sql_passwd: X6jx5nZe
sasl_sql_select: SELECT userpassword FROM user WHERE 
    username = '%u' AND 
    userrealm = '%r' AND 
    auth = '1'
sasl_sql_usessl: no
```

Testing II

Start both applications from separate terminals.
Let them choose the mechanisms!
Terminal 1

```
# sample-server -p 8000 -s rcmd
```

Terminal 2

```
# sample-client -p 8000 -s rcmd localhost
```
What makes the ldapdb-plugin special?

ldapdb is the most complex plugin currently available from the Cyrus SASL source tree:

- ldapdb uses proxy authentication
  The plugin must authenticate before it may authenticate the given data
- OpenLDAP expects SASL authentication
  The plugin must be configured to do SASL authentication
- SASL authentication must be configured for OpenLDAP
  slapd server
  OpenLDAP slapd must have been built to support SASL authentication

ldapdb-Plugin II

- slapd must only offer mechanisms the ldapdb-SASL-client can handle
- OpenLDAP does not permit a proxy-user to do proxy-authentication by default
  A global or a per-user policy must be configured
- OpenLDAP does not permit a proxy-user to search any path for proxy-authentication
  A search path must be configured.
Steps

- OpenLDAP
  - Directory Information Tree (DIT)
- slapd
  - basic configuration
  - SASL authentication
  - configure
  - test
  - Proxy-user
  - define search permissions
  - define search path
- ldapdb-Plugin
  - Understand parameters
  - configure sample-server
  - Test using sample-client and sample-server

Directory Information Tree
Basic configuration

Schema:

- include /etc/openldap/schema/core.schema
- include /etc/openldap/schema/cosine.schema
- include /etc/openldap/schema/inetorgperson.schema
- include /etc/openldap/schema/nis.schema

Database
- database hdb
- suffix "dc=example,dc=com"
- rootdn "cn=Manager,dc=example,dc=com"
- rootpw {CRYPT}Tv46kTMlpGuK.

Importing Directory Information Tree

Importing DIT offline

Your slapd(8) should not be running when you do this to ensure consistency of the database.

# /etc/init.d/ldap stop
# slapadd -v -c -b "dc=example,dc=com" -l example.com.ldif

Tip
Fix user and group permissions after the import...
Configuring Authentication Mapping

Users, using SASL authentication to login to OpenLDAP, are treated internally within a special context: The internal view either follows this “authentication request DN” pattern:

```
uid=<username>,cn=<realm>,cn=<mechanism>,cn=auth
```

or this one:

```
uid=<username>,cn=<mechanism>,cn=auth
```

Neither of these patterns matches the DN of the proxy-user!

Configuring Authentication Mapping II

An authentication mapping matches the authentication request DN against the proxy-user DN pattern:

```
authz-regexp
  uid=(.*)\,cn=.*\,cn=auth
  ldap://dc=example,dc=com??sub?\(|\(uniqueIdentifier=\$1\)\(mail=\$1\))
```

Important:

- More than one mapping may be configured
- First match wins!
Testing Authentication Mapping

- Use `ldapwhoami` as proxy-user to login to OpenLDAP
- Switch into role of user requesting authentication
- Show identity

```
# ldapwhoami -U proxyuser -X u:test@example.com -Y digest-md5
SASL/DIGEST-MD5 authentication started
Please enter your password: <proxyuser-Passwort>
SASL username: u:test@example.com
SASL SSF: 128
SASL installing layers
dn:cn=test,ou=people,dc=example,dc=com
Result: Success (0)
```
Proxy-User

Proxy-Authentication Policy
An authenticated proxy-user is by default not authorized to use other users’ credentials.

- policy in slapd.conf configures authorization
- policy is set using the authz-policy parameter

authz-policy parameter

Valid values (since OpenLDAP 2.3.x) are:

- to
  DN specifies destinations where proxy-user may use credentials
- from
  DN specifies a user permitted to act as proxy-user
- any
  Either policy may be used
- all
  Both policies must be given
Authorizing the Proxy-User

authz-policy parameter settings control which attribute must be added to user objects. Using to as authz-policy:

authzTo: dn.regex:uniqueIdentifier=(.*)\, ou=people, dc=example, dc=com

- Add authzTo-attribute to proxy-user object
- authzTo-attribute configures a LDAP search down the branch(es) where Proxy-User is authorized to authenticate.

Example

Using from as authz-policy:

authzFrom: dn.exact:uniqueIdentifier=proxyuser\, ou=auth, dc=example, dc=com

- A user adds the authzFrom attribute to its object, if she wants to authorize the proxy-user.
- The attribute defines the DN of the proxy-user that should be allowed to authenticate.
configuring ldapdb

ldapdb parameters

- auxprop_plugin: ldapdb
  The name of the LDAPDB-auxprop-plugin is ldapdb.

- ldapdb_uri
  Specifies one or more URIs (list) the plugin should use as authentication backend. Server may offer unencrypted (ldap://) or encrypted (ldaps://) connections.

- ldapdb_id
  Proxy-user username

- ldapdb_pw
  Proxy-user password in plaintext

configuring ldapdb II

- ldapdb_mech
  Specifies the mechanism the plugin should use when it logs into the LDAP server.

- ldapdb_rc
  Specifies a path to a configuration file where options for the ldapdb-LDAP-client would be stored. Such options could be paths to TLS certificates...

- ldapdb_starttls
  Specifies TLS requirement level (try or demand).
Testing ldapdb

Sample configuration (/usr/lib/sasl2/sample.conf):

log_level: 7
pwcheck_method: auxprop
auxprop_plugin: ldapdb
mech_list: PLAIN LOGIN DIGEST-MD5 CRAM-MD5
ldapdb_uri: ldap://localhost
ldapdb_id: proxyuser
ldapdb_pw: proxy_secret
ldapdb_mech: DIGEST-MD5

Testing ldapdb II

Both applications are run from different terminals:
Terminal 1

# sample-server -p 8000 -s rcmd -m PLAIN

Terminal 2

# sample-client -p 8000 -s rcmd -m PLAIN localhost
Postfix

configuration (/usr/lib/sasl2/smtpd.conf):

log_level: 7
pwcheck_method: auxprop
auxprop_plugin: ldapdb
mech_list: PLAIN LOGIN DIGEST-MD5 CRAM-MD5
ldapdb_uri: ldap://localhost
ldapdb_id: proxyuser
ldapdb_pw: proxy_secret
ldapdb_mech: DIGEST-MD5

Cyrus IMAP

configuration (/etc/imapd.conf):

sasl_log_level: 7
sasl_pwcheck_method: auxprop
sasl_auxprop_plugin: ldapdb
sasl_mech_list: PLAIN LOGIN DIGEST-MD5 CRAM-MD5
sasl_ldapdb_uri: ldap://localhost
sasl_ldapdb_id: proxyuser
sasl_ldapdb_pw: proxy_secret
sasl_ldapdb_mech: DIGEST-MD5
Security Considerations

Potential attacks
Network communication
Two areas where network communication may be eavesdropped:

- From client-application to server-application
  Use TLS to protect plaintext-mechanisms!
- From server-application to LDAP server
  Use secure mechanisms only

Credentials

Two areas where credentials can be eavesdropped:

- Client-application
  Protection depends on OS and client
- Server-application (ldapdb-plugin)
  Use TLS client certificate for ldapdb-plugin instead of a password!
Certification Authority

- Locations vary from distribution to distribution...
- Create CA
  - Use CA(.pl)-script to create CA
    ```
    # ./CA -newca
    ```
- We need certificates for OpenLDAP server and ldapdb-plugin.
  Important for proxy-user certificate:
  DN in proxy-user certificate must exactly match its DN in the directory!

Certification Authority II

Create request and key in one run:

```
# openssl req -new -nodes -keyout slapd_key.pem -out slapd_key.pem -days 365
```

Sign certificate:

```
# openssl ca -policy policy_anything -out slapd_cert.pem -infiles slapd_key.pem
```
Configuring slapd-Server

CA certificate, private key and public server certificate must be specified in `slapd.conf`:

```
TLSCACertificateFile /etc/pki/CA/cacert.pem
TLSCertificateFile     /etc/openldap/cacerts/slapd_cert.pem
TLSCertificateKeyFile  /etc/openldap/cacerts/slapd_key.pem
# Demand TLS while you test!
TLSVerifyClient demand
```

Configuring ldapdb-Client

```
/usr/lib/sasl2/smtpd.conf:

log_level: 7
pwcheck_method: auxprop
auxprop_plugin: ldapdb
mech_list: PLAIN LOGIN DIGEST-MD5 CRAM-MD5
ldapdb_uri: ldap://localhost
ldapdb_id: proxyuser
ldapdb_mech: EXTERNAL
ldapdb_starttls: demand
ldapdb_rc: /usr/lib/sasl2/ldaprc
```
Configuring ldapdb-Client II

/usr/lib/sasl2/ldaprc:

TLS_CERT /usr/lib/sasl2/ma_cert.pem
TLS_KEY /usr/lib/sasl2/ma_key.pem
TLS_CACERT /etc/pki/CA/cacert.pem
TLS_REQCERT demand

Questions?

Any questions?
Patrick Ben Koetter  
state of mind  
http://www.state-of-mind.de  
patrick.koetter@state-of-mind.de

Ralf Hildebrandt  
T-Systems  
http://www.arschkrebs.de  
ralf.hildebrandt@charite.de