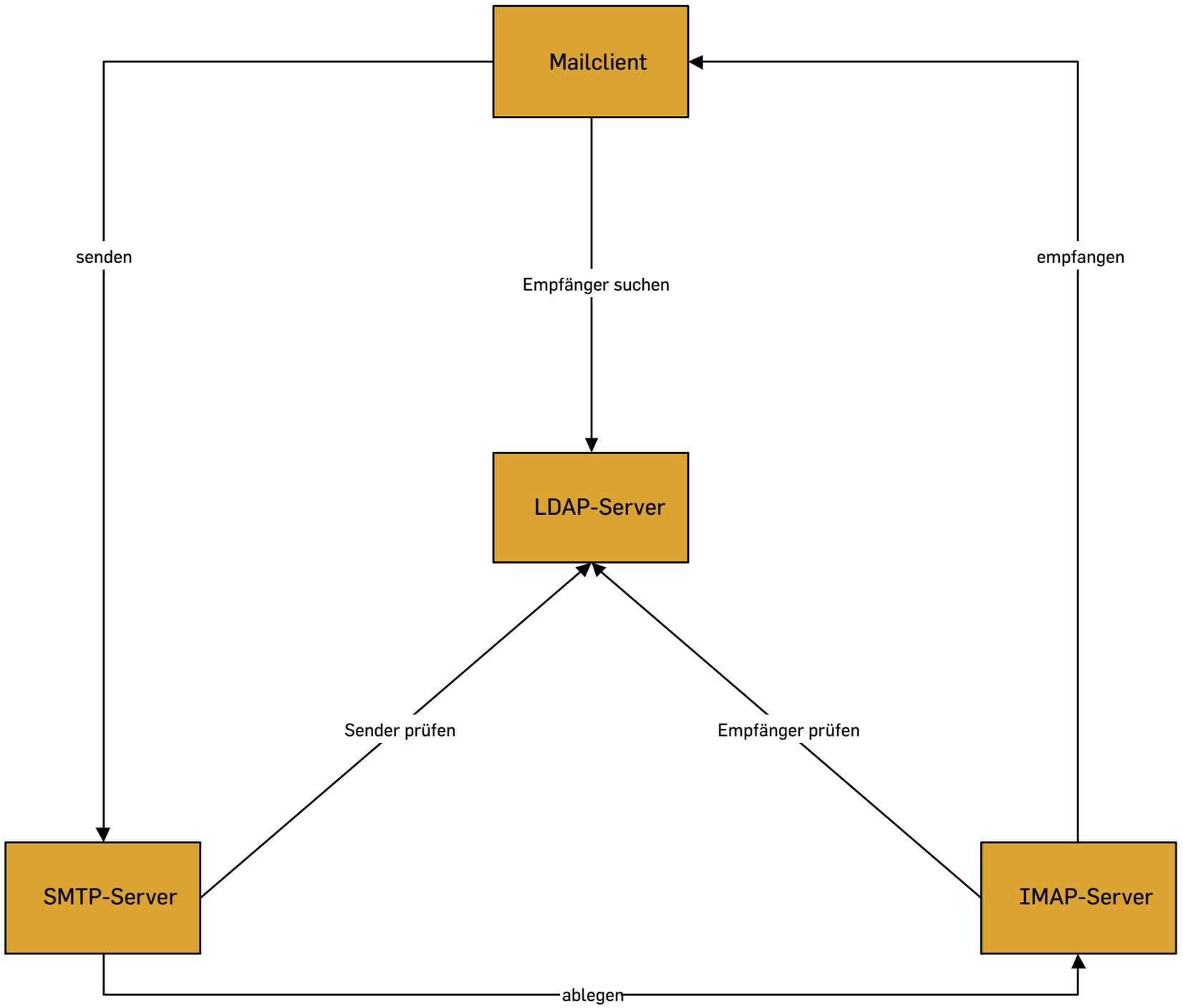
Authentifizierung mit Cyrus SASL

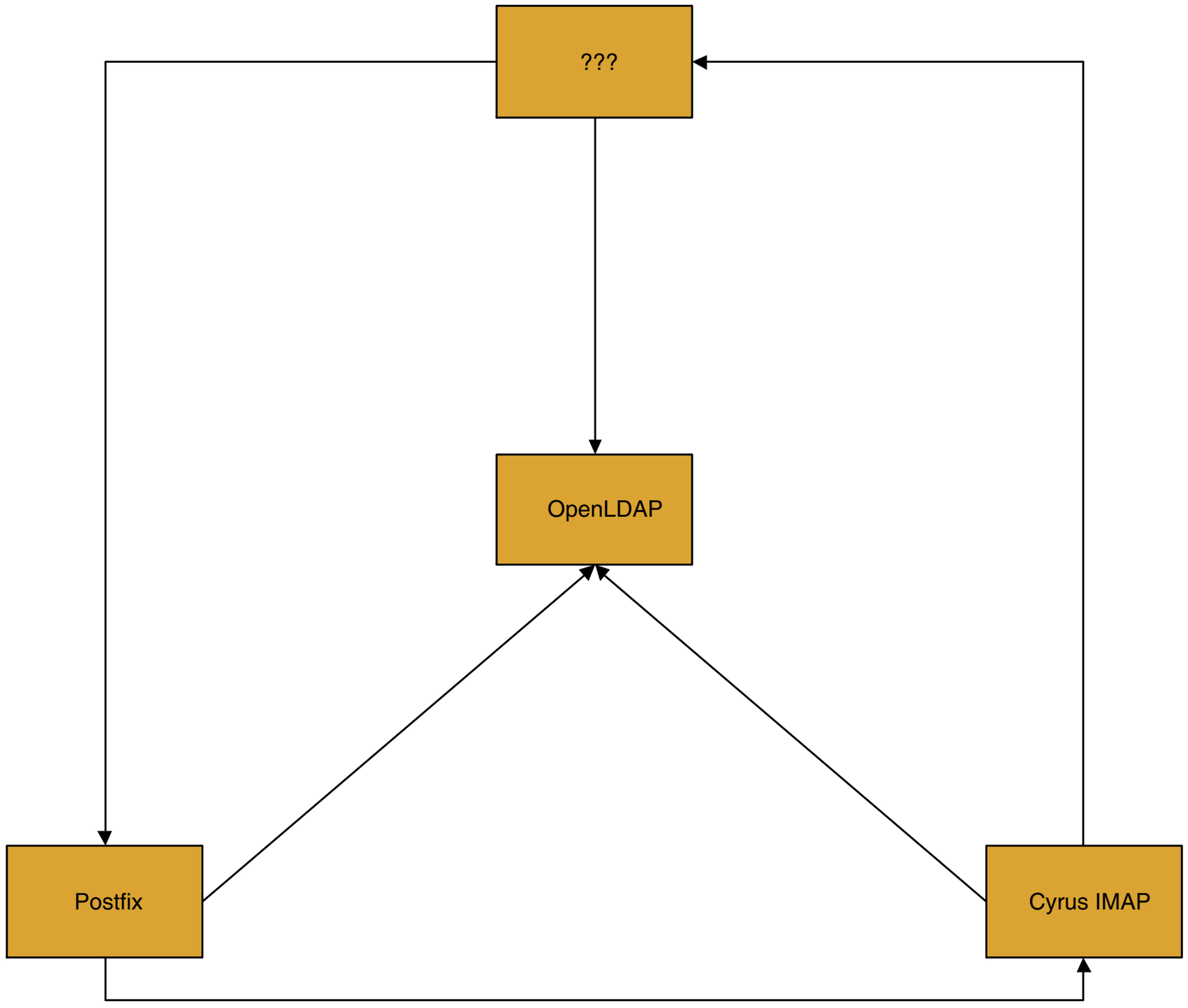
Szenario

Mailserver

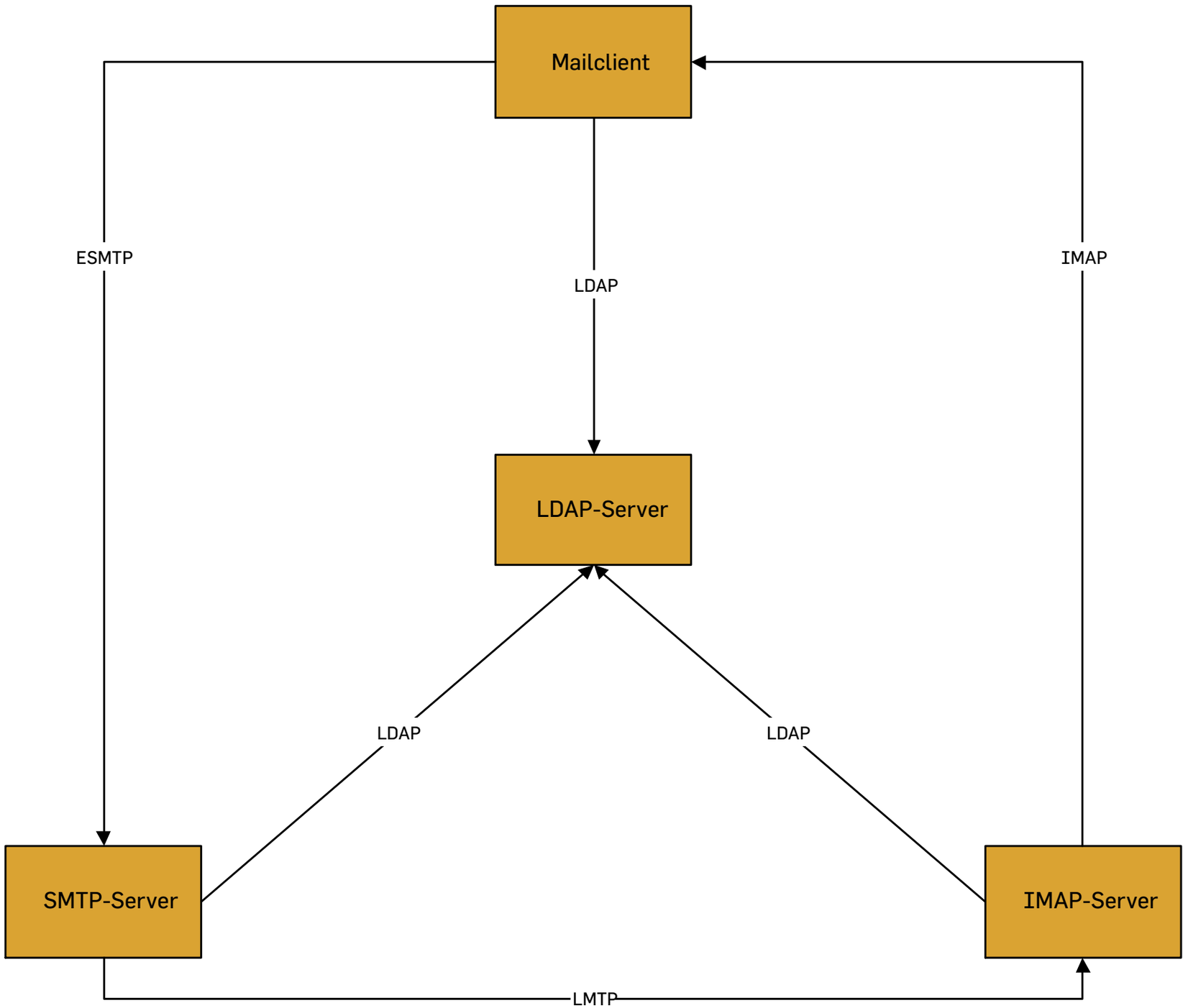


Architektur

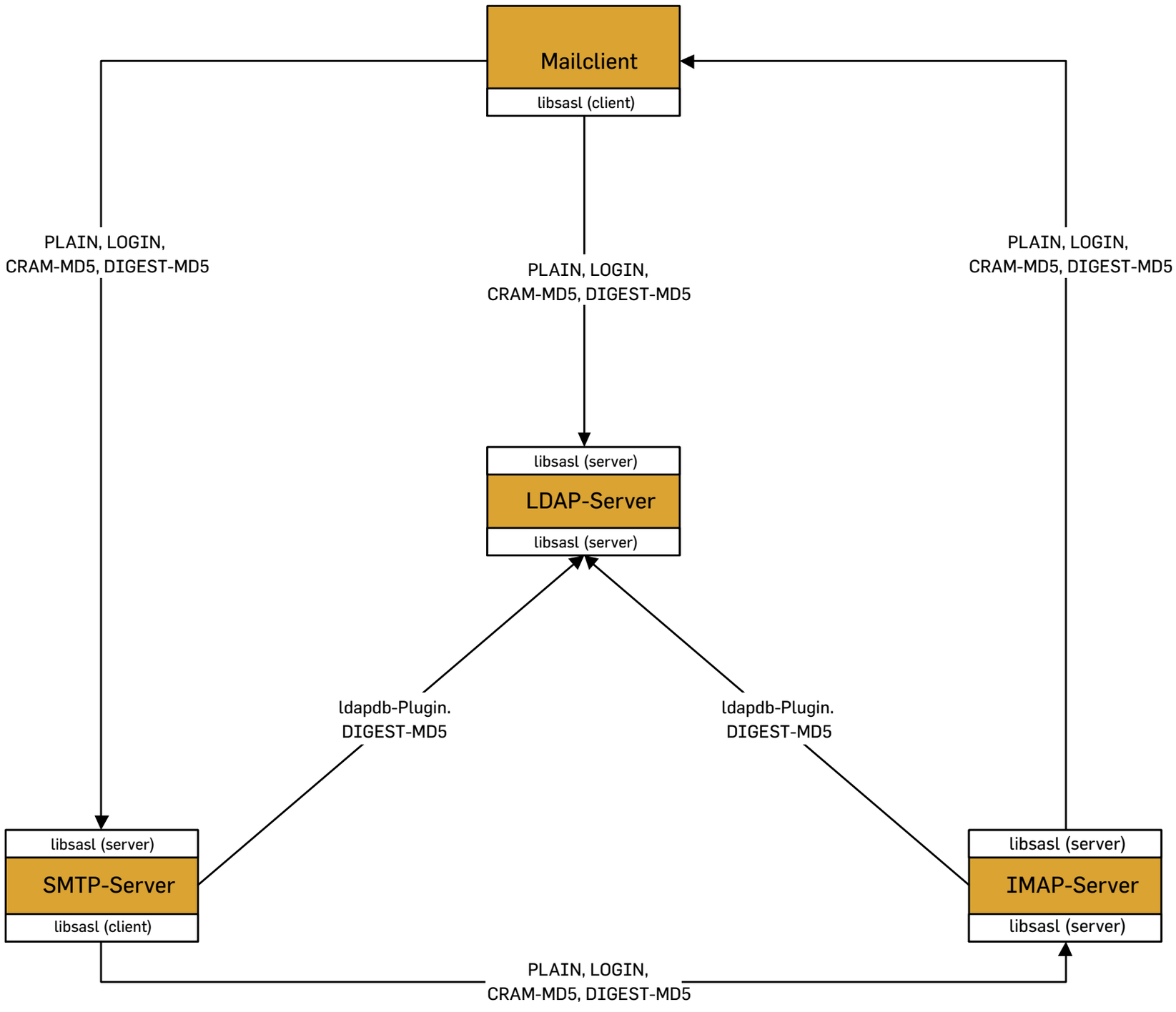
Software-Komponenten



Protokolle



Authentifizierung



Cyrus SASL

Aufgabe

Cyrus SASL ist

- ein Framework zur Anwender-Authentifizierung
- die Implementierung von SASL, dem Simple Authentication and Security Layer
- standardisiert und in RFC 2222 beschrieben
- „das Kind derer, die auf dem Standard hocken“...

Einsatz

Cyrus SASL agiert nicht für sich selbst, sondern es wird in Applikationen verbindungsorientierter Protokolle (z.B. SMTP, FTP, POP3, IMAP, LDAP) eingebettet.

„(...) a protocol 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.“

Vorteile

Wer Cyrus SASL in seine Applikationen integriert,

- vereinfacht die Software-Entwicklung
- kann sich auf stabile und erprobte Software verlassen
- erhöht die Interoperabilität seiner Applikation

Nachteile

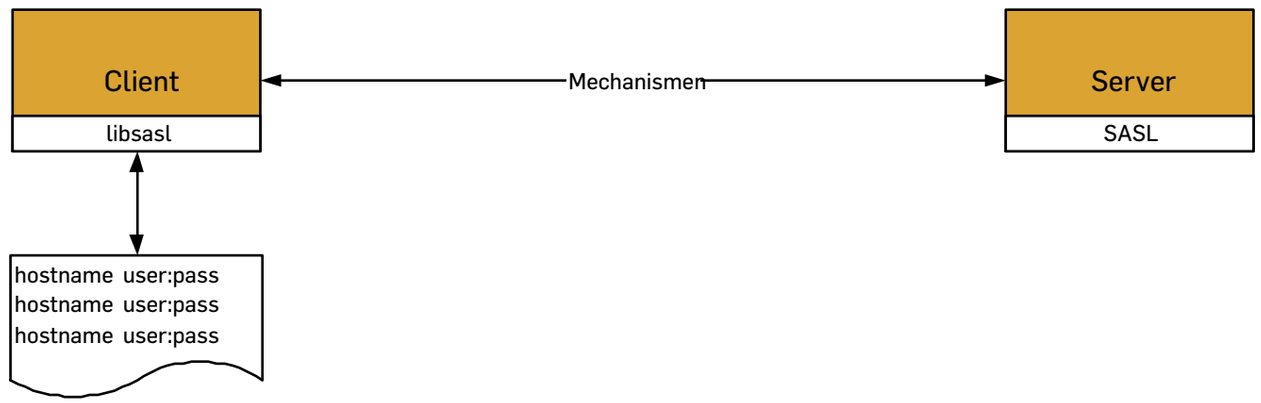
Wer Cyrus SASL in seine Applikationen integriert,

- treibt seine Anwender in den Wahnsinn, denn Cyrus SASL ist so gut wie nicht für Anwender dokumentiert!

Funktionsweise

- Cyrus SASL stellt Entwicklern eine library, `libsasl`, zur Verfügung
- libsasl wird vom Entwickler in seine Applikation eingebunden
- Je nach Modus der Applikation, Client- oder Server-Modus, nimmt `libsasl` unterschiedliche Aufgaben war

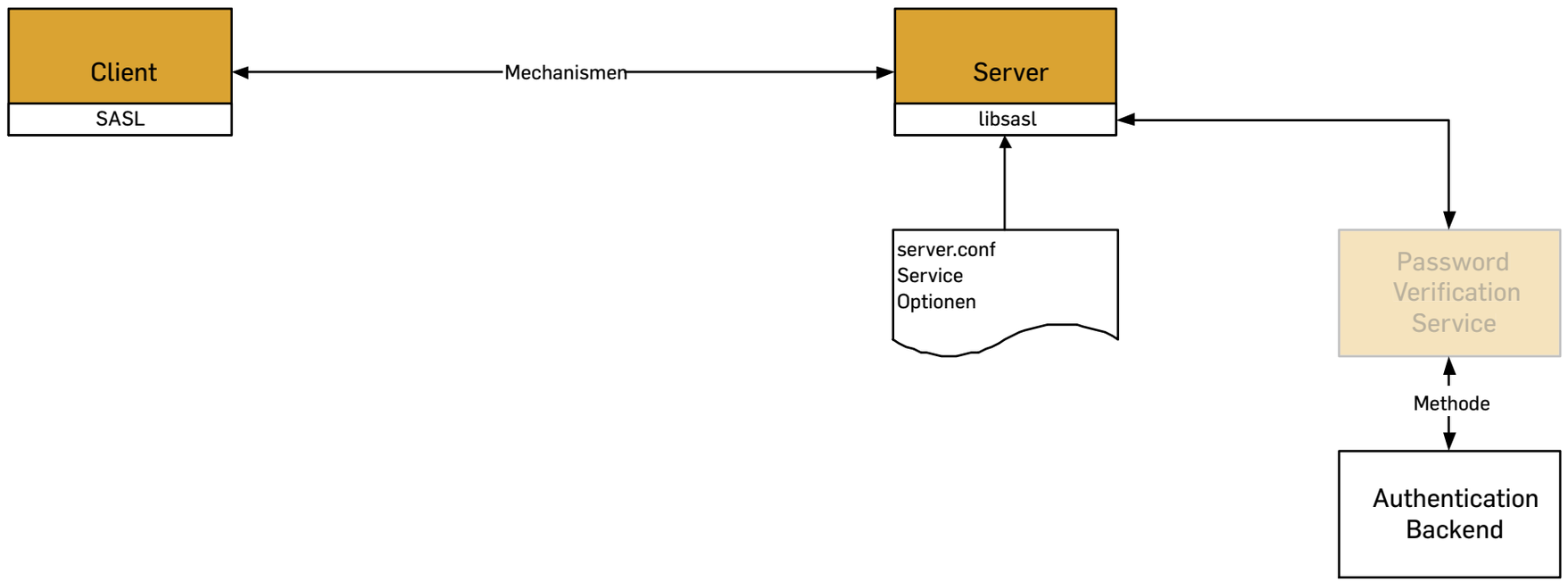
libsasl in Client-Applikation



Aufgaben

- bestimmt, welchen Mechanismus ein Client während einer Authentifizierung benutzen soll
- führt die Aufgaben des ausgewählten Mechanismus aus

libsasl in Server-Applikation



Aufgaben

- bestimmt, welche Mechanismen der Server anbieten kann
- führt die Aufgaben des gewählten Mechanismus aus
- übergibt Authentifizierungsdaten an eine Methode
- teilt dem Server das Ergebnis einer Authentifizierung mit

SASL-Begriffe im Überblick

- Client und Server kommunizieren über das ***Authentication Interface***
- Sie nutzen ***Mechanismen***, um Authentifizierungsdaten auszutauschen
- Eine ***Methode*** oder ein ***Password Verification Service*** greift auf ein ***Authentication Backend*** zu
- Die ***Methode authentifiziert*** die Authentifizierungsdaten.
- Der Server teilt dem Client das Ergebnis der Authentifizierung mit.
- Der ***Server autorisiert*** den Client gegebenenfalls, eine Aktion auszuführen

Authentication Interface

Authentication Interface ist die Schnittstelle an der sich Client und Server begegnen

- Client und Server-Kommunikation wird durch das verwendete Protokoll spezifiziert
- Als universelles Framework ***muss*** SASL frei von protokoll-spezifischen Anforderungen sein.
- Client- und Server-Kommandos für Authentifizierung werden deshalb von den Protokollen spezifiziert
- `libsasl` agiert als Bindeglied zwischen protokoll-spezifischen Kommandos und universellen SASL-Routinen

Mechanismen

Mechanismen sind Strategien zur Übermittlung von Authentifizierungsdaten.

> „SASL mechanism names must be registered with the IANA."

Unterschiede

- Vorgehensweise
  Wie wird bei der Authentifizierung vorgegangen?
- Daten
  Welche Daten werden zur Authentifizierung übermittelt?
- Sicherheit
  Welches Maß an Sicherheit bietet Kombination aus Vorgehensweise und übermittelten Daten?

Mechanismen-Gruppen

Mechanismen werden aufgrund ähnlicher Eigenschaften in Gruppen zusammengefaßt:

- Plaintext-Mechanismen
- Shared-Secret-Mechanismen
- Ticket-Mechanismen
- External-Mechanismen

Plaintext-Mechanismen

Vorgehensweise

Authentifizierungsdaten werden base64-kodiert, damit sie beim Transport nicht „kaputt“ gehen können.

Daten

Plaintext-Mechanismen übermitteln Benutzername, Password und ggf. Domain.

Sicherheit

- Transport erfolgt unverschlüsselt
  Transport-Schicht kann zum Schutz mit TLS verschlüsselt werden.
- Daten müssen auf Server vorgehalten werden

PLAIN

authcid, *authzid* und *password* werden zu einem String zusammengefaßt, base64-kodiert und als ein String übertragen

```
# perl -MMIME::Base64 -e ‚print encode_base64(„authcid\0authzid\
  0password“);‘
YXV0aGNpZABhdXRoemlkAHBhc3N3b3Jk
```

Beispiel (SMTP)

```
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 YXV0aGNpZABhdXRoemlkAHBhc3N3b3Jk
235 Authentication successful
QUIT
221 Bye
```

LOGIN

LOGIN ist ein proprietärer Mechanismus aus dem Hause Microsoft. Er ist nicht standardisiert und nicht frei dokumentiert.

Outlook und Outlook Express können kein PLAIN, aber dafür LOGIN.

Nutzlast des Mechanismus (Benutzername, Passwort und ggf. Domain) erinnert an den Windows Anmelde-Dialog

Die Daten werden meist (!) jedoch nacheinander übertragen.

Beispiel (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-Mechanismen

Vorgehensweise

Shared-Secret-Mechanismen sind Challenge-Response-Verfahren.

Der Server stellt eine Aufgabe (Challenge), die der Client nur lösen kann (Response), wenn er über dieselben Authentifizierungsdaten verfügt.

Daten

- Benutzername und Geheimnis (Challenge) mit Hilfe des Passwortes verschlüsselt.
- Der gesamte String wird base64-kodiert übertragen.
- Das Passwort wird nie gesendet.

Sicherheit

- Transport erfolgt kodiert und verschlüsselt
- Daten müssen auf Server vorgehalten werden

Verfügbare Mechanismen

- CRAM-MD5
- DIGEST-MD5
- NTLM

External-Mechanismen

EXTERNAL ist ein externer, nicht in SASL befindlicher Mechanismus.

> „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.“

SASL verläßt sich also darauf, dass der externe Mechanismus die Authentifizierung durchgeführt hat.

TLS

TLS ist einziger EXTERNAL-Mechanismus, der hin und wieder „in the wild“ angetroffen wird.

- TLS bietet Client- und Server-Authentifizierung über Zertifikate.
- Die Transport-Schicht ist, dank TLS, verschlüsselt.

Ticket-Mechanismen

Vorgehensweise

- User authentifiziert sich bei Kerberos-Server und erhält ein Ticket.
- Mit dem Ticket kann der User nun Zugang zu weiteren Diensten anfordern.

Daten

- Client (User) übermittelt anfangs mit Benutzername und Passwort an Kerberos-Server.
- Client sendet nur noch Ticket an Server für Zugang.

Sicherheit

- Weder Benutzername noch Passwort werden später von Client zum Server übertragen

Verfügbare Mechanismen

Cyrus SASL unterstützt

- Kerberos_4
  sollte nicht verwendet werden, weil verwundbar
- GSSAPI (Kerberos_5)
  gilt als „der“ sichere Mechanismus schlechthin, ist allerdings ein Thema für ein eigenes Seminar

Macht Postfix smtp-Client Sinn?

```
Jun 27 21:35:58 mailer2 postfix/smtp[3668]: > mailer1.uml.local[10.0.0.2]: AUTH GSSAPI
YIIB+wYJKoZIhvcSAQICAQBuggHqMIIB5qADAgEFoQMCAQ6iBwMFACAAAACjggEKYYIBBjCCAQKgAwIBBaEL
GwlVTUwuTE9DQUyiJDAioAMCAQOhGzAZGwRzbXRwGxFtYWlsZXIxLnVtbC5sb2NhbKOBxzCBxKADAgEQoQM
CAQOigbcEgbQakXOfyFibOLfHuM+fdaRMJLHLXIQ3M33B/LIvyEAcNjbukcGgFaLgCvI19zy2+EbNf1Usnt
cE7j9b1UQFcNhgpaubKJOtpsJ50I7SNfoAvFwd7fb6UokvboP/qTwGS2miIGGfY+O/Vdm77CvPBBJ3SxgTW
WDnYrFdNQ6leYyVWjsKff6e8YRBqtezOgV0QCtvcGDGnAEMlZqVn1dufi6oT0wIdz5+ZfzWyYzFikxEWKX6
WOkgcIwgb+gAwIBEKKBtwSBtH/0ZFECCy7JCel/670uE9ZvNN0y5rEOSf/P0AGmho43X/pmEn4ntFgBACiCtQj
gAA2dYxAI1c2gf0yDRnKXthYNKbauescFcotb16E/MK6zT/QEmT8e1bZBAcK2xXKWzciL7il8VXjNiF30bqMs/
DABfBqcTsIZ/mFLI0iFCKHU6mxwEu32GnjAMVakcSEurQl04wamUIRIUu+qODM8WKRtMYcXMP9K7pYKpSa4uptpQ
B0wWQ==

Jun 27 21:35:58 mailer2 postfix/smtp[3668]: < mailer1.uml.local[10.0.0.2]: 334 YIGWBgkqh
kiG9xIBAgICAG+BhjCBg6ADAgEFoQMCAQ+idzB1oAMCARCibgRskJhRl7eSu5DtkIEzvo2AMX8CVc2VCKv4rGEI

Jun 27 21:35:58 mailer2 postfix/smtp[3668]: smtp_sasl_authenticate: mailer1.uml.
local[10.0.0.2]: decoded challenge: `????*?H???????

Jun 27 21:35:58 mailer2 postfix/smtp[3668]: > mailer1.uml.local[10.0.0.2]:
Jun 27 21:35:58 mailer2 postfix/smtp[3668]: < mailer1.uml.local[10.0.0.2]: 334 YD8GCSqGS
Ib3EgECAgIBBAD/////MMYEJPLlkELHld2xVbaKmOQlzV/RsQO7YXFGS0gjzca9IkETAQAAAAQEBAQ=

Jun 27 21:35:58 mailer2 postfix/smtp[3668]: smtp_sasl_authenticate: mailer1.uml.
local[10.0.0.2]: decoded challenge: `???*?H??????????

Jun 27 21:35:58 mailer2 postfix/smtp[3668]: smtp_sasl_authenticate: mailer1.uml.
local[10.0.0.2]: uncoded client response `???*?H??????????

Jun 27 21:35:58 mailer2 postfix/smtp[3668]: > mailer1.uml.local[10.0.0.2]: YD8GCSqGSIb3E
gECAgIBBAD/////Zhk8ATiz2AruYJJMNOEqwHklJajQHh0HwXdmg4hXdZTGULGfAQAAAHADAwM=

Jun 27 21:35:58 mailer2 postfix/smtp[3668]: < mailer1.uml.local[10.0.0.2]: 235
Authentication successful
```

Methoden

Methoden authentifizieren im Auftrag von libsasl Benutzerdaten.

Welche gibt es?

Cyrus SASL kennt zwei Arten von Methoden:

- Password Verification Services
- auxprop-Plugins

Password Verification Service

Password Verification Services

- werden als Dienste (daemon) auf einem Rechner ausgeführt
- können mit besonderen Privilegien (!) ausgeführt werden
- können auf besonders geschützte authentication backends zugreifen
- beherrschen bisher nur die „unsicheren“ Plaintext-Mechanismen

Welche gibt es?

Cyrus SASL kennt drei Password Verification Services

- pwcheck
- saslauthd
- authdaemond

pwcheck

- pwcheck ist der alte, ursprüngliche Password Verification Service von Cyrus SASL
- Er war bis Ende Version 1.5.xx im Einsatz
- Er ist heute noch Teil der Cyrus SASL Sourcen
- pwcheck gilt längst als deprecated

saslauthd

saslauthd ist der offizielle, aktuelle Password Verification Service von Cyrus SASL

saslauthd kann auf verschiedenste authentication backends zugreifen

```
# saslauthd -v
saslauthd 2.1.19
authentication mechanisms: getpwent kerberos5 pam rimap shadow ldap
```

- getpwent
  Direkter Zugriff auf passwd-Datei
- kerberos5
  Gegen lokalen Kerberos realm authentifizieren
- pam
  Anfrage an Pluggable Authentication Modules (PAM) senden und Ergebnis benutzen
- rimap
  Login mit Benutzerdaten bei remote IMAP-Server versuchen.
- shadow
  Zugriff auf shadow-Datei
- ldap
  Authentifizierung (einfacher BIND) gegen LDAP-Server

authdaemond

authdaemond ist ein Password Verfication Service, der auf den Courier authdaemond Authentifizierungsdienst zugreift.

authdaemond kann wie alle Password Verifications Services nur Plaintext-Mechanismen verarbeiten.

Alle authentication backends, die authdaemond erreichen kann, stehen zur Verfügung:

- authuserdb
  Eigene Datenbank des Courier MTA Projektes anfragen
- authpam
  Anfrage an Pluggable Authentication Modules (PAM) senden und Ergebnis benutzen
- authpgsql
  PostgreSQL-Datenbank abfragen
- authldap
  Authentifizierung (einfacher BIND) gegen LDAP-Server
- authmysql
  MySQL-Datenbank abfragen
- authcustom
  Eigene Implementierung schreiben...

auxprop-Plugins

auxprop-Plugins können mehr als die Daemonen. Laut Spezifikation können sie

- Benutzerdaten authentisieren
- Benutzerkonten erstellen
- Benutzerkonten (Passwort) verwalten
- Benutzerdaten umschreiben (canon_user_plugins)

Welche auxprop-Plugins gibt es?

Cyrus SASL liefert in Version 2.1.22 folgende auxprop-Plugins mit den Sourcen aus:

- sasldb
- sql
- ldapdb

sasldb

sasldb ist das Standard authentication backend von Cyrus SASL

- sasldb ist eine Berkeley DB
- Formatänderung von Cyrus SASL 1.x zu 2.x, damit auch Shared-Secret-Mechanismen mit sasldb angeboten werden können
- seit Cyrus SASL 2.x werden Passwörter deshalb im Klartext abgespeichert

Utilities

- `saslpasswd2`
  - Anlegen einer sasldb2
  - Anlegen von Benutzerkonten
  - Modifizieren von Benutzerkonten
- `sasldblistusers2`
  Auflisten von Benutzern einer sasldb2

sql

sql ist ein generischer Treiber für den Zugriff auf verschiedene SQL-Server

- MySQL
- PostgreSQL
- SQLite

Viele Admins ziehen es vor, SQL-Anbindung über PAM herzustellen, damit sie die Passworte in der SQL-DB verschlüsselt ablegen können. Benutzername und Passwort lassen sie allerdings unverschlüsselt vom Client zum Server senden...

Ein patch („Frost“-Patch) für das auxprop-Plugin sql ermöglicht Zugriff auf verschlüsselte Passworte, verhindert aber die Nutzung von Shared-Secret-Mechanismen.

ldapdb

ldapdb ist ein Treiber für den Zugriff auf einen OpenLDAP-Server

Das ldapdb-Plugin realisiert den Zugriff als Proxy-User wie ein RFC 2222 beschrieben:

> „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."

ldapdb erfordert Cyrus SASL Authentifizierung zweimal zu konfigurieren:

- Cyrus SASL Login an slapd
- ldapdb-Plugin

Konfiguration

Was muss konfiguriert werden?

Client

- Alles was ein Client benötigt sind Benutzerdaten
- Clients müssen nicht, können aber konfiguriert werden

Server

Server-Applikationen müssen konfiguriert werden. Es gibt ***zwei Arten*** libsasl SASL Konfigurationen mitzuteilen:

- `application_name`
  Der `application_name` definiert die Konfigurationsdatei nach der libsasl suchen wird (z.B. Postfix smtpd.conf-Datei)
- Werte werden beim Initialisieren der libsasl übergeben (z.B. Cyrus IMAP)

Die Server-Applikation übergibt zusätzlich an `libsasl`

- `service_name`
  Der `service_name` definiert den Service (Protokoll) den libsasl bedienen soll. Dieser Wert ist u.a. für PAM wichtig, damit es weiß, wie der Name der richtigen PAM-Konfigurationsdatei lautet

Parameter

Cyrus SASL kennt generische Parameter und methoden-spezifische Parameter.

Methoden-spezifische Parameter werden

- bei Password Verification Services als Kommandozeilen-Optionen übergeben
- bei auxprop-Plugins in der `*.conf`-Datei der Server-Applikation notiert

Generische Parameter

`log_level`

Der log_level-Parameter definiert den Detailgrad der Log-Nachrichten an den syslogd-Log-dienst.

Level	Detailgrad
0	Keine Protokollierung
1	Ungewöhnliche Fehler protokollieren
2	Alle Authentifizierungsfehler protokollieren
3	Warnungen, die nicht verhängnisvolle Fehler nennen, protokollieren
4	Ausführlicher als 3 protokollieren
5	Ausführlicher als 4 protokollieren
6	Spuren interner Protokollabläufe protokollie-ren
7	Spuren interner Protokollabläufe und Kenn-worte protokollieren

Logging ist Glückssache...

Kein Password Verification Service oder auxprop-Plugin realisiert alle Stufen; manche reali-sieren überhaupt kein Logging!

`pwcheck_method`

Der pwcheck_method-Parameter definiert einen oder mehrere (!) Password Verification Services oder auxprop-Plugins, die zur Authentifizierung herangezogen werden sollen.

Gültige Werte sind die Namen der Password Verification Services oder auxprop-Plugins.

`mech_list`

Der mech_list-Parameter definiert jene Mechanismen, die eine Server-Applikation ihren Clients anbieten darf.

Gültige Werte sind die Namen der Mechanismen durch whitespace voneinander getrennt.

Methoden-spezifische Parameter

... sehen wir uns nacher im Detail an.

Testen

Test-Werkzeuge

Testen von Cyrus SASL ist wichtig, damit getrennt werden kann, ob ein Konfigurationsfehler in Cyrus SASL oder in der (Server-) Applikation vorliegt.

Viele Admins verbringen Stunden damit, den Fehler in der Applikation zu suchen...

Problem

Cyrus SASL hat keine „richtigen“ Test-Werkzeuge!

testsaslauthd

testsaslauthd kann nur den Password Verification Service saslauthd testen

Problem

Ein erfolgreicher Test mit testsaslauthd ist kein Beweis, dass das Cyrus SASL Framework komplett funktioniert, denn testsaslauthd benutzt nicht (!) die libraries der Cyrus SASL Mechanismen...

Kommando

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

client — server

Cyrus SASL liefert Sample-Applikationen für Entwickler, um die Integration der library zu demonstrieren.

Überraschung!

Die Sample-Applikationen sind nicht dokumentiert...

Server

```
# ./sample-server -h
lt-sample-server: Usage: lt-sample-server [-b min=N,max=N] [-e ssf=N,id=ID] [-m MECH] [-
f FLAGS] [-i local=IP,remote=IP] [-p PATH] [-d DOM] [-u DOM] [-s NAME]

-b ...  #bits to use for encryption
        min=N   minumum #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-seperated 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
lt-sample-client: Usage: lt-sample-client [-b min=N,max=N] [-e ssf=N,id=ID] [-m MECH] [-
f FLAGS] [-i local=IP,remote=IP] [-p PATH] [-s NAME] [-n FQDN] [-u ID] [-a ID]

        -b ...  #bits to use for encryption
                min=N   minumum #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-seperated search path for mechanisms
-r REALM        realm to use    -s NAME service name 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
```

Beispiele

shadow-Authentifizierung

Arbeitsschritte

- saslauthd-Umfeld vorbereiten
- test-User anlegen
- Testen
  - mit testsaslauthd
  - mit sample-server und sample-client
- AUTH einbinden
  - 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             Enable verbose logging
  -h             Display this message.
```

saslauthd-Umfeld vorbereiten

Klassiker

Das Socket-Directory (`run_path`) fehlt...

```
# /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.
```

Testen

Test-User anlegen

```
# useradd test
# passwd test
```

Testen mit testsaslauthd

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

Testen mir sample-server und sample-client

sample-server übermittelt `sample` als `application_name`.

/usr/lib/sasl2/sample.conf

```
pwcheck_method: saslauthd
mech_list: PLAIN LOGIN
```

Zum Testen werden beide sample-Applikationen in unterschiedlichen Terminals aufgerufen:

Terminal 1

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

Terminal 2

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

AUTH einbinden

Cyrus SASL erlaubt zwei Vorgehensweisen zur Konfiguration applikationsspezifischer Einstellungen:

- separate Datei in `/usr/lib/sasl2`
  demnächst konfigurierbares Config-Dir
- Übergabe der Parameter durch applikationseigene Konfigurationsdatei an `libsasl`

Postfix

Postfix benutzt separate Konfigurationsdatei und übermittelt per default den `application_name` `smtpd` an `libsasl`.

/usr/lib/sasl2/smtpd.conf

```
pwcheck_method: saslauthd
mech_list: PLAIN LOGIN
```

Cyrus IMAP

Cyrus IMAP übergibt alle Werte an `libsasl` aus seiner zentralen Konfigurationsdatei.

/etc/imapd.conf

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

sasldb-Authentifizierung

Arbeitsschritte

- sasldb2 erstellen
- Testen mit sample-server und sample-client
- AUTH einbinden
  - in Postfix
  - in Cyrus IMAP

saslpasswd2

```
# 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] [-
f sasldb] [-u DOM] userid

        -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
```

sasldb erstellen

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

sasldb-Inhalt auflisten

```
# sasldblistusers2 -h

This product includes software developed by Computing Services
at Carnegie Mellon University (http://www.cmu.edu/computing/).

sasldblistusers2: usage: sasldblistusers2 [-v] [[-f] sasldb]
        -f sasldb       use given file as sasldb
        -v      print version numbers and exit
```

```
# sasldblistusers2
test@example.com: userPassword
```

Testen

sample-server übermittelt `sample` als `application_name`.

/usr/lib/sasl2/sample.conf

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

Zum Testen werden beide sample-Applikationen in unterschiedlichen Terminals aufgerufen:

Terminal 1

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

Terminal 2

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

AUTH einbinden

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
```

ldapdb-Plugin

Was macht das ldapdb-Plugin besonders?

Das ldapdb-Plugin ist das komplexeste auxprop-Plugin, das Cyrus SASL im Augenblick kennt:

- ldapdb führt Proxy-Authentifizierung durch
  Das Plugin muss sich vor der eigentlichen Authentifzierung erst selbst am LDAP-Server anmelden!
- OpenLDAP erwartet SASL-Authentifizierung
  Das Plugin das sich anmeldet, muss selbst SASL-Authentifizierung beherrschen
- SASL-Authentifizierung muss im OpenLDAP-Server slapd konfiguriert sein
  Der OpenLDAP-Server muss SASL-Authentifizierung beherrschen
- slapd darf nur Mechanismen anbieten mit denen der ldapdb-SASL-Client umgehen kann
  SASL für den slapd-Server muss konfiguriert werden
- OpenLDAP gestattet generell nicht, dass der Proxy-User Proxy-Authentifizierung durchführt
  Eine globale oder per-User Policy muss eingepflegt werden
- OpenLDAP gestattet nicht, dass der Proxy-User überall Proxy-Authentifizierung durchführt
  Ein Such-Bereich muss konfiguriert werden

Aufgaben-Liste

OpenLDAP

slapd

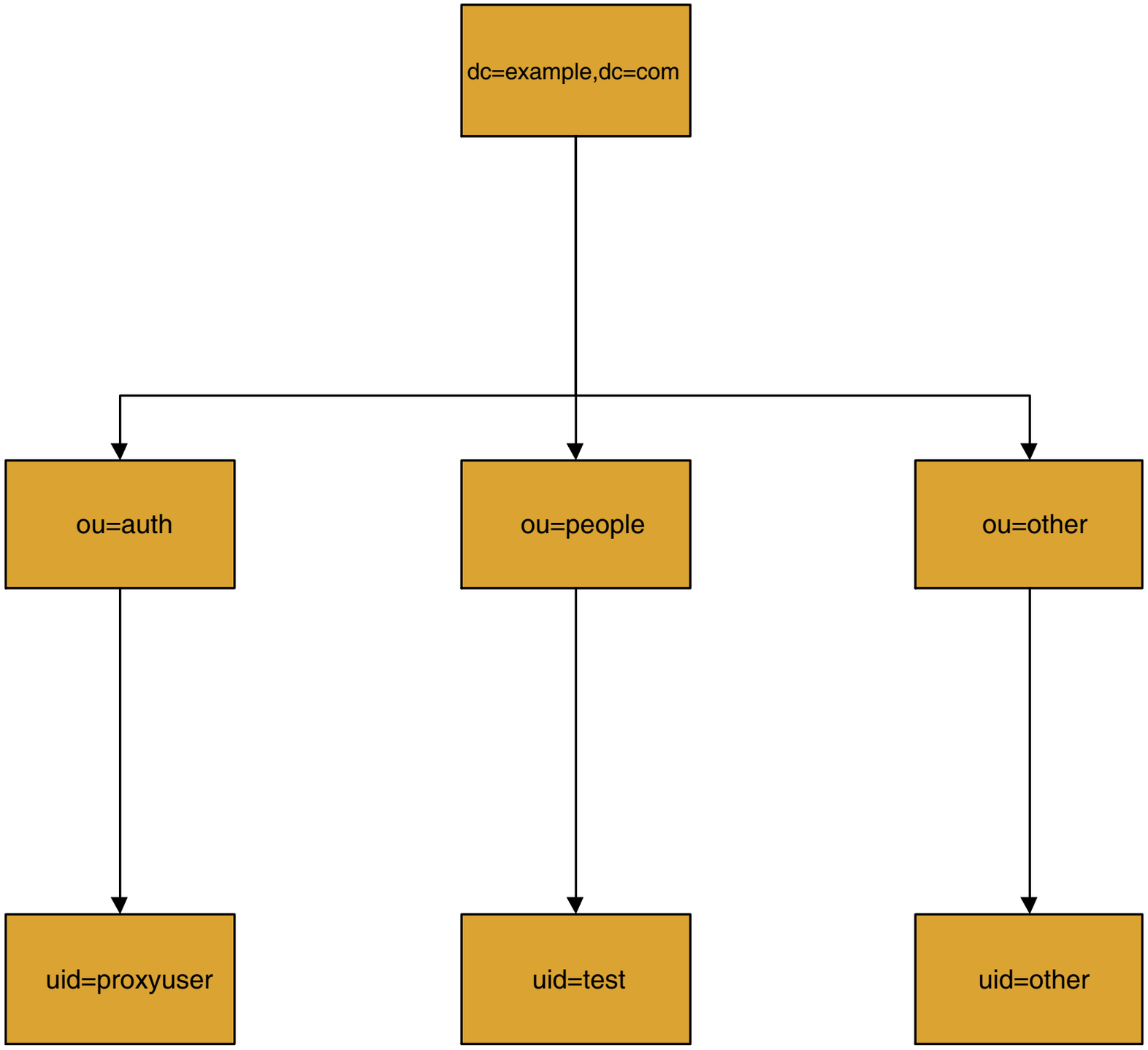
- Basiskonfiguration
- DIT importieren
- SASL-Authentifizierung
  - konfigurieren
  - testen
- Proxy-User
  - Such-Recht bestimmen
  - Such-Bereich festlegen

ldapdb-Plugin

- Parameter im Detail
- sample-server konfigurieren
- Testen mit sample-client und sample-server

DIT

Aufbau



slapd

Basiskonfiguration

Schemata

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

Datenbank

```
database        bdb
suffix          „dc=example,dc=com“
rootdn          „cn=Manager,dc=example,dc=com“
rootpw          {CRYPT}Tv46kTM1pGuK.
```

DIT importieren

DIT offline importieren

„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

User- und Gruppenrechte auf neue Dateien anpassen.

SASL-Authentifizierung

Authentication Mapping konfigurieren

User, die sich per SASL-Authentifizierung an OpenLDAP anmelden, werden intern in einem besonderen Kontext dargestellt.

Die Darstellung folgt einem beider „authentication request DN“-Muster:

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

oder

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

Dieses Muster „paßt“ nicht auf den DN unseres Proxy-Users!

Wir müssen ein mapping erstellen, das einen authentication request DN so auf unsere Struktur abbildet, dass unser Proxy-User gefunden wird:

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

Wichtig

- Es können mehrere mappings konfiguriert werden.
- Es gilt: First match wins!

Authentication Mapping testen

Mit `ldapwhoami` als Proxy-User an den Server anmelden, in die Rolle des zu authentifizierenden Users wechseln und danach die Identität ausgeben lassen.

```
# 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

Policy für Autorisierung

Ein ***authentifizierter*** Proxy-User ist ***per default nicht autorisiert*** die Anmeldedaten anderer Benutzer zur Authentifizierung zu nutzen.

- Eine zentrale Policy in `slapd.conf` legt die Autorisierung für Proxy-User fest.
- Die Policy wird mit dem `authz-policy`-Parameter definiert.

authz-policy-Parameter für Proxy-User

Gültige Werte (ab OpenLDAP 2.3.x) sind:

`to`
: DN legt global fest für wen (branch) er zur Authentifizierung autorisiert ist

`from`
: DN legt fest einzeln (object) fest, wer ihn autorisiert Authentifizierung durchzuführen

`any`
: Eine der beiden Autorisierungen muss gegeben sein

`all`
: Beide Autorisierungen müssen gegeben sein

Entry autorisieren

Die Autorisierungspolicy bestimmt das Attribut, das an User-Entries angefügt werden muss.

to als authz-policy

- Der Proxy-User wird mit dem `authzTo`-Attribut ausgestattet.
- Das Attribut definiert eine LDAP-Suche für den Branch in dem der Proxy-User für eine Authentifizierung autorisiert ist.

Eintrag für DIT:

```
authzTo:    ldap:///ou=people,dc=example,dc=com??sub? \
    (&(objectclass=inetOrgPerson)(mail=*))
```

from als authz-policy

- Ein User, der authentifiziert werden können möchte, stattet seinen Entry mit dem authz-From-Attribut aus.
- Das Attribut definiert den DN, der autorisiert ist, eine Proxy-Authentifizierung durchzuführen.

Möglicher Eintrag für Seminar-DIT:

```
authzFrom:    dn.exact:uid=proxyuser,ou=auth,dc=example,dc=com
```

ldapdb-Plugin konfigurieren

ldapdb-Parameter

`auxprop_plugin: ldapdb`
: Der Name des LDAPDB-auxprop-Plugins ist `ldapdb`.

`ldapdb_uri`
: Ein oder mehrere URIs (Liste), die auf zu verwendende LDAP-Server verweisen. LDAP-Server können unverschlüsselte (ldap://) oder verschlüsselte (ldaps://) Verbindungen anbieten.

`ldapdb_id`
: Benutzername des Proxy-Users

`ldapdb_pw`
: Passwort des Proxy-Users in Plaintext

`ldapdb_mech`
: Mechanimus den das ldpadb-Plugin benutzen soll, wenn es sich als Proxy-User an den LDAP-Server anmeldet.

`ldapdb_rc`
: Pfad zu einer Konfigurationsdatei für den ldapdb-LDAP-Client. Dort können LDAP-Client spezifische Parameter (z.B. Zertifikate für TLS) gesetzt werden.

`ldapdb_starttls`
: Vorgaben, ob TLS nur versucht („`try`“) werden soll oder Voraussetzung („`demand`“) für Durchführung der Anfrage ist.

ldapdb testen

Test-Konfiguration

/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
```

Zum Testen werden beide sample-Applikationen in unterschiedlichen Terminals aufgerufen:

Terminal 1

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

Terminal 2

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

Postfix

Konfiguration

/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

Konfiguration

/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
```

Sicherheit

Angriffstellen

Netzwerk-Kommunikation

Netzwerk-Kommunikation der Authentifizierung kann an zwei Streckenabschnitten angegriffen werden:

- Client-Applikation zu Server-Applikation
  Plaintext-Mechanismen mit TLS schützen
- Server-Applikation zu LDAP-Server
  Nur sichere Mechanismen einsetzen

Anmeldedaten

Anmeldedaten für Authentifizierung können an zwei Streckenabschnitten angegriffen werden:

- Client-Applikation
  Schutz-Mechanismen sind OS- und Client-abhängig
- Server-Applikation (ldapdb-Plugin)
  ldapdb-Plugin kann anstatt Passwort ein TLS-Client-Zertifikat verwenden!

Certification Authority

Ablageort der PKI-Infrastruktur variiert von Distribution zu Distribution.

CA erstellen

`CA.pl`-Skript oder nur `CA`-Skript dient dazu eine CA zu erstellen

```
# ./CA -newca
```

Wir benötigen Zertifikate für den OpenLDAP-Server und für das ldapdb-Plugin.

Wichtig für Proxy-User-Zertifikat

Der DN des Proxy-User-Zertifikates muss identisch sein mit seinem DN im Directory!

Request und Key in einem Zertifikat (key) erstellen

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

Zertifikat durch Signatur von CA erstellen

```
# openssl ca -policy policy_anything -out slapd_cert.pem \
  -infiles slapd_key.pem
```

slapd-Server-Zertifikat

CA-Zertifikat, privater Schlüssel und öffentliches Zertifikat müssen in `slapd.conf` eingebunden werden.

```
TLSCACertificateFile  /etc/pki/CA/cacert.pem
TLSCertificateFile    /etc/openldap/cacerts/slapd_cert.pem
TLSCertificateKeyFile /etc/openldap/cacerts/slapd_key.pem
```

Für den Testlauf fordern wir explizit TLS!

```
TLSVerifyClient       demand
```

ldapdb-Client-Zertifikat

/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
```

/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
```

Fragen?

Speaker

state of mind
Patrick Koetter
patrick.koetter@state-of-mind.de

Download

http://postfix.state-of-mind.de

Copyright