Simple Authentication and Security Layer

oder

Cyrus SASL im Detail

Patrick Ben Koetter

2. Mailserver Konferenz

19. Mai 2005, Magdeburg

EINLEITUNG

Mit SMTP Authentifizierung kann Postfix „fremden“ Clients das Relayen erlauben und die Nutzung bestimmter envelope sender autorisieren. Postfix kann SMTP AUTH auch nutzen, um sich selber bei einem entfernten Server zu autorisieren.

Langjährige Erfahrungen auf der deutschen und englischen Mailingliste zeigen:

- Viele postmaster haben Probleme SMTP AUTH für Postfix zu konfigurieren.
- Nahezu ausnahmslos scheitern die postmaster an der Konfiguration der dazu notwendigen Software Cyrus SASL.

ÜBERSICHT

- Cyrus SASL
  - Architektur
  - Layer
  - Planung
  - Konfiguration
  - Testen
- Postfix SMTP AUTH Konfiguration
  - Serverseitiges SMTP AUTH
  - Clientseitiges SMTP AUTH
- Zukunft von Cyrus SASL in Postfix

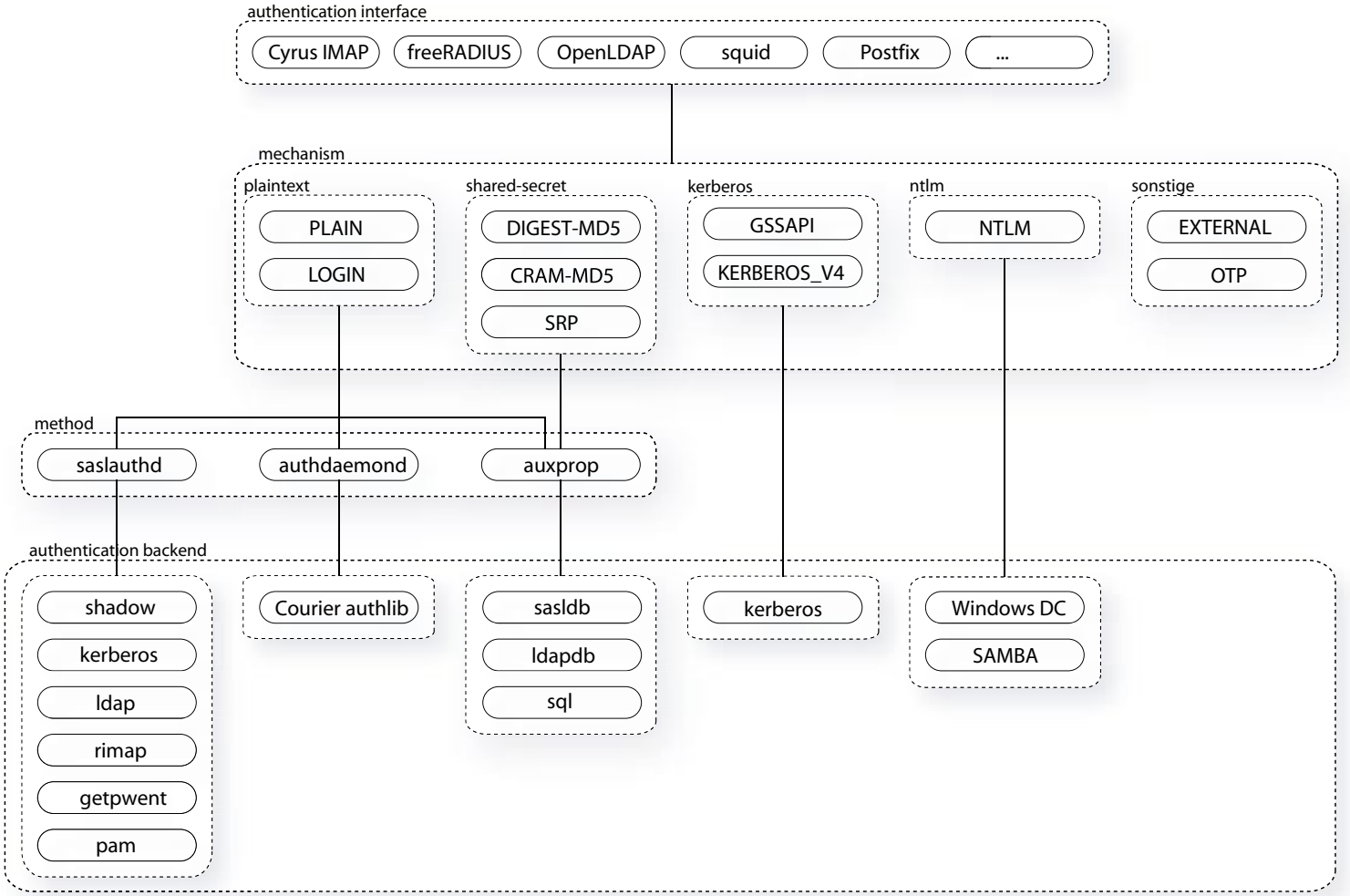
SIMPLE AUTHENTICATION AND SECURITY LAYER

SASL ist ein Authentication Framework. Es stellt Funktionen zur Verfügung, damit Applikationen, die Authentifizierung benötigen, diese nicht selber implementieren müssen.

Das Framework besteht aus drei Layern; ein Password Verification Service vermittelt zwischen diesen Layern:

Authentication Interface
Mechanism
Method

The Big Picture



Authentifizierungsablauf

1. Server bietet Client `AUTH` und einige **Mechanismen** an.
2. Client wählt Mechanismus aus und berechnet entsprechend des Mechanismus den Authentifizierungsstring.
3. Client meldet AUTH an den Server und übergibt entsprechend des Mechanismus Authentifzierungsdaten.
4. Server übergibt Daten an libsasl
5. **Passwort verification service** kontaktiert mittels passender **method** das **authentication backend** und versucht Daten zu verifizieren.
6. Ergebnis der Verifikation wird an Server mitgeteilt.
7. Server kann Client für bestimmte Aktionen autorisieren.

AUTHENTICATION INTERFACE

SASL überläßt es deshalb dem jeweiligen Dienst, das authentication interface zu integrieren.

Beispiel: SMTP Dialog als authentication interface

```
# telnet mail.example.com 25
220 mail.example.com ESMTP Postfix
EHLO client.example.com
250-mail.example.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-AUTH PLAIN LOGIN CRAM-MD5 DIGEST-MD5
250-AUTH=PLAIN LOGIN CRAM-MD5 DIGEST-MD5
250 8BITMIME
QUIT
```

MECHANISM

Mechanismen sind standardisierte Vorgehensweisen, um Authentifizierungsdaten zu übertragen.

Cyrus SASL kennt folgende Klassen von Authentifzierungsmechanismen:

- PLAINTEXT
- SHARED-SECRET
- KERBEROS
- EXTERNAL
- OTP
- SRP
- ...

plaintext

Plaintext Mechanismen übertragen Benutzername, Kennwort und ggf. auch den REALM vom Client zum Server als base64 encodierten string.

base64 unsicher, denn es läßt sich einfachst decodieren!

```
$ perl -MMIME::Base64 -e ‚print decode_base64(„dGVzdAB0ZXN0AHRlc3RwYXNz“);‘
testtesttestpass
```

PLAIN

PLAIN ist der offene plaintext Standard.

LOGIN

LOGIN berücksichtigt zusätzlich proprietäre Vorgaben für Microsoft Mail Clients

Beispiel: plaintext Authentifizierung

```
220 mail.example.com ESMTP Postfix
EHLO example.com
250-mail.example.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-AUTH DIGEST-MD5 CRAM-MD5 GSSAPI PLAIN LOGIN
250-AUTH=DIGEST-MD5 CRAM-MD5 GSSAPI PLAIN LOGIN
250-XVERP
250 8BITMIME
AUTH PLAIN dGVzdAB0ZXN0AHRlc3RwYXNz
235 Authentication successful
QUIT
```

Muß plaintext angeboten werden, dann SMTP Kommunikation durch TLS vor Unbeteiligten absichern!

shared-secret

server überträgt ein Geheimnis (challenge) – errechnet auf Basis von Nutzerdaten (Benutzername, Kennwort, ggf. REALM) – an client.

client errechnet Auflösung (response) auf Basis seiner eigenen Nutzerdaten.

Ist Lösung korrekt, stimmen die Nutzerdaten beider überein.

CRAM-MD5
    ist der originäre shared-secret Mechanismus.
DIGEST-MD5
    ist der sichere Nachfolger.

METHOD

Methods (oder: Password Verification Services) stellen eine Verbindung zwischen den mit Mechanismen übertragenen Authentifizierungsdaten und authentication backends her.

Authentication backends sind Datenquellen in denen Benutzernamen und Kennworte (ggf. REALM und andere Informationen) vorgehalten werden.

Cyrus SASL (Version 2.1.21) verfügt über drei Methods:

– saslauthd
- auxprop
- authdaemond

saslauthd

Standalone daemon, der über einen lokalen socket kommuniziert und auf plaintext Mechanismen beschränkt ist. Nachfolger von pwcheck der 1.x Versionen.

Verfügbare authentication backends:

- POSIX Accounts (passwd, shadow)
- PAM
- Datenbanken (LDAP)
- IMAP Server
- Kerberos Server

Server übergibt Autentifizierungsdaten an saslauthd; saslauthd antwortet mit `success` oder `fail`.

auxprop

Kurzform für auxilliary property plugins, bezeichnet eine Anzahl von authentication backends repräsentiert durch libraries.

Server, der auxprop nutzt, kümmert sich selber um die Abwicklung der Authentifizierung.

Generell keine Beschränkung bei Mechanismen.

Verfügbare auxilliary property plugins in 2.1.21:

- sasldb
- sql
- ldapdb

authdaemond

Delegation der Authentifizierung an den authdaemond daemon (Courier authlib). Dieser password verification service beherrscht wie saslauthd ausschließlich plaintext Mechanismen.

Zugriff auf alle authentication backends, die authlib nutzt:

- LDAP
- MySQL
- PostgreSQL
- eigene Implementierungen

Vereinfachung der Konfiguration und Administration eines Gesamtsystems auf Kosten der Sicherheit!

CYRUS SASL PLANUNG

- Welche Mechanismen müssen angeboten werden?
- In welchem authentication backend werden die Authentifizierungsdaten vorgehalten?

Wahl der Mechanismen

Die AUTH Fähigkeiten der Clients bestimmen welche Mechanismen angeboten werden müssen.

Überblick über AUTH Fähigkeiten bei Cyrus SASL Developer Alexeji Melnikov unter http://www.melnikov.ca/mel/devel/SASL_ClientRef.html.

Im nächsten Schritt erfolgt Reduktion auf die sichersten Mechanismen.

Sind plaintext Mechanismen weiterhin mit von der Partie, sollte TLS zum Schutz der Nutzerdaten eingesetzt werden!

Authentication Backends

Genauere Betrachtung folgender authentication backends:

- POSIX Accounts
- PAM
- IMAP
- LDAP
- LDAPDB
- SASLDB
- SQL

POSIX Accounts

saslauthd kann als `root` ausgeführt werden und damit auf `/etc/passwd` oder `/etc/shadow` lesend zugreifen.

- Beide backends sind bereits vorhanden
- Tools zur Erstellung, Modifikation und Löschung von Accounts bestehen ebenfalls

Aber:

- User in diesem authentication backend haben typischerweise SHELL Zugang.
- Sensible Daten (`root`) sind ebenfalls enthalten.

Von Gebrauch in Firmen- oder ISP-Umfeld ist abzuraten!

PAM

saslauthd kann Authentifzierung an PAM delegieren.

- Erweiterung um alle für PAM verfügbaren authentication backends

Aber:

- PAM ist nicht trivial
- PAM ist immer noch als DRAFT <http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html> dokumentiert

Wer sich mit PAM auskennt, kommt sicherlich weit. Wer es nicht kennt, sollte die Finger davon lassen, besonders dann wenn SASL selber in der Lage ist, das gewünschte authentication backend z.B. SQL oder LDAP direkt anzusprechen.

IMAP

Bei einer IMAP gestützten Authentifizierung versucht Cyrus SASL einen Login bei einem IMAP Server.

Die Authentifizierungsdaten werden beim Login von Cyrus SASL unverschlüsselt übertragen! Ohne Vorkehrungen keine sichere Lösung!

Wenn beide Server (SMTP, IMAP) auf demselben Host einsetzt werden, SASL besser auf dieselbe userdatenbank wie die des IMAP Servers zugreifen lassen kann.

Wenn das nicht geht, zur Not auf Kommunikation über das loopback Interface ausweichen.

LDAP

saslauthd kann auf LDAP Server lesend zugreifen.

Konfiguration ist komplex, weil LDAP komplex ist.

saslauthd

prüft nur, ob die strings für username und password übereinstimmen; Authentifizierung ist auf plaintext Mechanismen beschränkt.

Bedeutsamkeit der Daten im LDAP server sollte bestimmen, ob es mit plaintext allein getan ist!

LDAPDB

Das ldapdb auxprop plugin führt einen bind() Versuch mittels der vom Client übermittelten Daten durch.

ldapdb ermöglicht die Verwendung von shared-secret Mechanismen.

Zugriff auf LDAP server und seine Daten ist damit besser geschützt.

sasldb

sasldb macht besonders Sinn mit Cyrus IMAP, benützt aber die Berkeley DB und ist damit auch den Versionsinkompatibilitäten der Berkeley DB ausgesetzt.

Kennwörter werden in der sasldb unverschlüsselt abgelegt! sasldb muß restriktiv vor lokalen Zugriffen geschützt werden.

Auf Kommandozeile können mit `saslpasswd2` die, in sasldb enthaltenen, User administriert werden.

sasldb ist unpraktisch in größeren Umgebungen, da ein (Web)Frontend zur Kennwortänderung für Enduser fehlt.

SQL

Einheitliche Parameter für Lese- und Schreibzugriffe (!) auf folgende SQL Server:

- MySQL
- PostgreSQL
- SQLite

plugin beherrscht nur unverschlüsselte Kennwörter in SQL DB.

Patches, die verschlüsselte Kennwörte in DB möglich machen, existieren:

- selten aktuell
- eher ein hack als eine saubere Integration

Wer diese Patches nutzt und seine DB entsprechend verschlüsselt anlegt, der stellt sich im schlimmsten Fall bei einem Security Update von Cyrus SASL vor die Frage, ob der Exploit in Kauf genommen werden muß oder ob „schnell mal“ die User DB ins Unverschlüsselte migriert wird.

CYRUS SASL KONFIGURATION

- Welcher password verification service ?
- Welche Mechanismen dürfen angeboten werden?
- Wie soll auf authentication backends zugegriffen werden?

Name der Konfigurationsdatei

Cyrus SASL wählt anhand des vom Server übergebenen `applicationname` die passende Konfigurationsdatei aus.

Der Name der Konfigurationsdatei setzt sich aus ***Applikationsname*** und Suffix ***conf*** zusammen.

In Postfix ist `smtpd` der Server, der mit Cyrus SASL in Kontakt tritt. `smtpd` übermittelt als `applicationname` seinen eigenen Namen.

Der Name der Konfigurationsdatei lautet also per default ***smtpd.conf***; seit Postfix 2.1. ist der Name mit `smtpd_sasl_application_name` konfigurierbar.

Globale Parameter

Folgende Parameter sind immer notwendig:

- `log_level`
- `pwcheck_method`
- `mech_list`

`log_level`

Cyrus SASL loggt mit syslog facilitiy `auth.*`.

```
auth.*                          -/var/log/auth.log
```

Loglevel sind je nach method lückenhaft implementiert:

0 No logging.
1 Log unusual errors. This is the default.
2 Log all authentication failures.
3 Log non-fatal warnings.
4 More verbose than 3.
5 More verbose than 4.
6 Log traces of internal protocols.
7 Log traces of internal protocols, including passwords.

`pwcheck_method`

`pwcheck_method` legt fest welcher Password verification service die Aufgabe der Authentisierung übernehmen soll.

Gültige Werte beim gegenwärtigen Entwicklungsstand von Cyrus SASL sind:

- `saslauthd`
- `auxprop`
- `authdaemond`

Veraltet:

- `pwcheck`

`mech_list`

Welche Mechanismen darf der Server den Clients anbieten?

- PLAIN
- LOGIN
- CRAM-MD5
- DIGEST-MD5

Seltene SASL Mechanismen für SMTP sind:

- GSSAPI
- KERBEROS
- OTP
- NTLM

Der client bestimmt den zu nutzenden Mechanismus!

Basiskonfiguration einer smtpd.conf

saslauthd

```
# Global parameters
log_level: 3
pwcheck_method: saslauthd
mech_list: PLAIN LOGIN
```

auxprop

```
# Global parameters
log_level: 3
pwcheck_method: auxprop
mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5
```

authdaemond

```
# Global parameters
log_level: 3
pwcheck_method: authdaemond
mech_list: PLAIN LOGIN
```

`mech_list` immer an die Fähigkeiten des password verification service anpassen. Clients wählen sonst einen starken Mechanismus und scheitern, weil der password verification service den Mechanismus nicht verarbeiten kann!

saslauthd

saslauthd benötigt ein Verzeichnis für seinen socket.

Beispiel: Fehlendes Verzeichnis

```
# saslauthd -d -a shadow
saslauthd[6503] :main            : num_procs  : 5
saslauthd[6503] :main            : mech_option: NULL
saslauthd[6503] :main            : run_path   : /var/state/saslauthd
saslauthd[6503] :main            : auth_mech  : shadow
saslauthd[6503] :main            : could not chdir to: /var/state/saslauthd
saslauthd[6503] :main            : chdir: No such file or directory
saslauthd[6503] :main            : Check to make sure the directory exists and is
saslauthd[6503] :main            : writeable by the user this process runs as.
```

Verzeichnis entweder anlegen - mindestens saslauthd und User `postfix` müssen darauf zugreifen können - oder saslauthd mit der Option `-m` in ein anderes Verzeichnis lenken:

Beispiel: Socket in anderem Verzeichis als default

```
# saslauthd -m /var/run/saslauthd -a shadow -d
saslauthd[6515] :main            : num_procs  : 5
saslauthd[6515] :main            : mech_option: NULL
saslauthd[6515] :main            : run_path   : /var/run/saslauthd
saslauthd[6515] :main            : auth_mech  : shadow
saslauthd[6515] :ipc_init        : using accept lock file: /var/run/saslauthd/mux.accept
saslauthd[6515] :detach_tty      : master pid is: 0
saslauthd[6515] :ipc_init        : listening on socket: /var/run/saslauthd/mux
saslauthd[6515] :main            : using process model
saslauthd[6515] :have_baby       : forked child: 6516
saslauthd[6515] :have_baby       : forked child: 6517
saslauthd[6515] :have_baby       : forked child: 6518
saslauthd[6515] :have_baby       : forked child: 6519
```

saslauthd authentication backends bestimmen

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

„authentication mechanisms“ heissen seit langem „authentication backends“!

getpwent

getpwent ermöglicht den Zugriff auf die passwd Datei für Systeme, die keine shadow Datei haben:

```
# saslauthd -m /var/run/saslauthd -a getpwent
```

pam

Wird pam angegeben wendet sich saslauthd an die pluggable authentication modules. PAM selbst erwartet bei Postfix als SMTP server eine `smtp.conf` in z.B. `/etc/pam.d/`:

```
# saslauthd -m /var/run/saslauthd -a pam
```

rimap

Mit rimap kontaktiert saslauthd einen Remote IMAP server zur Authentifzierung. Mit der Option `-O` muß angegeben werden, an welchen IMAP server sich saslauthd hierbei wenden soll:

```
# saslauthd -m /var/run/saslauthd -a rimap -O localhost
```

shadow

shadow ermöglicht den Zugriff auf die shadow Datei. saslauthd wird hierzu wie folgt gestartet:

```
# saslauthd -m /var/run/saslauthd -a shadow
```

ldap

Die Angabe weiterer Informationen (hostname des LDAP servers, query syntax usw.) ist notwendig.

Schwierig alles über Kommandozeile anzugeben! LDAP Konfiguration wird in einer `saslauthd.conf` abgelegt.

Standardpfad ist `/usr/local/etc/saslauthd.conf`. Anderer Pfad kann mit Option `-O` gesetzt werden:

```
# saslauthd -m /var/run/saslauthd -a ldap -O /etc/saslauthd.conf
```

Beispiel: `saslauthd.conf`

```
ldap_servers: ldap://127.0.0.1/ ldap://172.16.10.7/
ldap_bind_dn: cn=saslauthd,dc=example,dc=com
ldap_bind_pw: anothersecret
ldap_timeout: 10
ldap_time_limit: 10
ldap_scope: sub
ldap_search_base: dc=people,dc=example,dc=com
ldap_auth_method: bind
ldap_filter: (|(&(cn=%u)(&(uid=%u@%r)(smtpAuth=Y)))
ldap_debug: 0
ldap_verbose: off
ldap_ssl: no
ldap_start_tls: no
ldap_referrals: yes
```

Beschreibung der Parameter unter https://bugzilla.andrew.cmu.edu/cgi-bin/cvsweb.cgi/src/sasl/saslauthd/LDAP_SASLAUTHD?rev=1.7&content-type=text/x-cvsweb-markup

auxprop

authentication backend wird mit `auxprop_plugin` Parameter bestimmt.

```
# Global parameters
log_level: 3
pwcheck_method: auxprop
mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5

# auxiliary Plugin parameters
auxprop_plugin: sasldb
```

Wird kein oder ein falsches `auxprop_plugin` angegeben, wird `sasldb` als default benutzt.

Bei falscher Konfiguration irreführende Fehlermeldungen!

sasldb

Der Pfad zu `sasldb2` (default: `/etc/sasldb2`) kann mit `sasldb_path` individuell konfiguriert werden.

```
# Global parameters
log_level: 3
pwcheck_method: auxprop
mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5

# auxiliary Plugin parameters
auxprop_plugin: sasldb
sasldb_path:    /etc/cyrus-sasl/sasl.db
```

sql

```
# Global parameters
log_level: 3
pwcheck_method: auxprop
mech_list: PLAIN LOGIN CRAM-MD5 DIGEST-MD5

# auxiliary Plugin parameters
auxprop_plugin: sql
sql_engine: mysql
sql_hostnames: localhost
sql_database: mail
sql_user: postfix
sql_passwd: Yanggt!
sql_select: SELECT %p FROM users WHERE username = ,%u‘ \
    AND userrealm = ,%r‘ AND auth = ,1‘
sql_usessl: no
```

sql_engine Parameter gibt, um welchen SQL server es sich handelt.

Als Platzhalter für die Queries dienen folgende macros:

`%u` Platzhalter wird ersetzt mit `username`.

`%p` Platzhalter wird ersetzt mit `password`.

`%r` Platzhalter wird ersetzt mit `realm`.

`%v` Platzhalter der für einen Wert, der einen bestehenden Wert überschreiben soll, z.B. während SQL `%UPDATE` or `INSERT` Operationen.

ldapdb

ldapdb authentisiert den Nutzer in `smtpd.conf` für OpenLDAP. Dieser Nutzer kann, wenn in OpenLDAP Authorisierung vorliegt, einen `re-bind()` mit Nutzerdaten des clients machen.

```
# Global parameters
log_level: 7
pwcheck_method: auxprop
mech_list: PLAIN LOGIN DIGEST-MD5 CRAM-MD5

# auxiliary Plugin parameters
auxprop_plugin: ldapdb
ldapdb_uri: ldap://mail.example.com
ldapdb_id: proxyuser
ldapdb_pw: proxy_secret
ldapdb_mech: DIGEST-MD5
```

Der Suchpfad selbst wird in OpenLDAP konfiguriert.

Beispiel: slapd.conf

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

authdaemond

authdaemond erwartet die Angabe des Pfades zum socket des authdaemond servers.

```
# Global parameters
log_level: 7
pwcheck_method: authdaemond
mech_list: PLAIN LOGIN

# auxiliary Plugin parameters
authdaemond_path: /usr/lib/courier-imap/var/authdaemon/socket
```

Das Verzeichnis des sockets muß Zugriff für den User `postfix` gestatten!

CYRUS SASL TESTEN

Inkonsistenter Parametergebrauch und unterschiedlichste Anforderungen der authentication backends machen es schwierig, Cyrus SASL auf Anhieb richtig zu konfigurieren.

Die Meisten testen Authentifizierung falsch, indem sie sofort SMTP AUTH anstatt Cyrus SASL zuerst separat testen!

Es fehlen dedizierte, dokumentierte und valide Testwerkzeuge.

Zum Glück gibt es Beispielapplikationen, die zum Testen miß-braucht werden können.

testsaslauthd

Ein utility um saslauthd basierte Authentifizierung zu testen.

Gelungende Authentifizierung mit `testsaslauthd` nicht valide, denn `testsaslauthd` benützt nicht die libraries von Cyrus SASL, sondern interne Routinen!

Sinnvoller ist es daher, bevorzugt die Sample-Applikationen `server` und `client` zu benutzen!

server

`server` ist der Server einer beispielhaften SASL Implementierung der Cyrus SASL Sourcen.

`server` überträgt den ***Applikationsnamen*** `sample` an SASL. Konfigurationsdatei muss daher `sample.conf` heißen.

Wer die Konfiguration eines anderen Servers testen will, spart die Tipparbeit und vermeidet mögliche Fehler, indem die notwendige `sample.conf` auf beispielsweise `smtpd.conf` gelinkt wird:

```
# cd /usr/lib/sasl2/
# ln -s smtpd.conf sample.conf
```

`server` wird mit Angabe eines Services und eines freien Ports gestartet.

```
# server -s rcmd -p 8000
trying 10, 1, 6
socket: Address family not supported by protocol
trying 2, 1, 6
```

client

`client` erwartet, zusätzlich zur Angabe desselben Services und Ports wie `server`, den zu benutzenden Mechanismus und den `hostname` auf dem `server` auf eingehende Verbindungen lauscht.

```
# client -s rcmd -p 8000 -m PLAIN 127.0.0.1
receiving capability list... recv: {11}
PLAIN LOGIN
PLAIN LOGIN
please enter an authentication id: test
please enter an authorization id: test
Password:
send: {5}
PLAIN
send: {1}
Y
send: {18}
test[0]test[0]testpass
successful authentication
closing connection
```

POSTFIX SMTP AUTH KONFIGURATION

Serverseitiges SMTP AUTH

Parameter werden im Regelfall global in `main.cf` gesetzt.

SMTP AUTH serverseitig aktivieren:

```
smtpd_sasl_auth_enable = yes
```

Mechanismen nach Sicherheitsgrad filtern:

```
smtpd_sasl_security_options = noanonymous
```

Nicht RFC-konforme clients unterstützen:

```
broken_sasl_auth_clients = yes
```

Authentifizierten Clients das Relayen gestatten:

```
smtpd_recipient_restrictions =
    ...
    permit_sasl_authenticated
    permit_mynetworks
    ...
```

Optionale Parameter

Per default einen REALM anhängen, wenn keiner übergeben wird:

```
smtpd_sasl_local_domain = example.com
```

Netzwerken oder Hosts kein SMTP AUTH anbieten:

```
smtpd_sasl_exceptions_networks = 172.16.0.0/24, 10.0.0.1/32
```

Envelope Sender mit SMTP AUTH kontrollieren

Postfix kann mit SMTP AUTH die Nutzung bestimmter envelope sender Adressen autorisieren oder verbieten.

Eine map, beispielsweise `/etc/postfix/sender_login`, bildet envelope sender auf SASL Login Namen ab:

```
# envelope sender                SASL Login Name
info@example.com                 barney@example.com, wilma@example.com
postmaster@example.com           bammbamm@example.com
barney@example.com               barney@example.com
wilma@example.com                wilma@example.com
```

Die map wird in `main.cf` eingebunden:

```
smtpd_sender_login_maps = hash:/etc/postfix/sender_login
```

Restriktionen überprüfen, ob die Vorgaben der map eingehalten werden:

```
smtpd_recipient_restrictions =
    ...
    reject_unauthenticated_sender_login_mismatch
    reject_sender_login_mismatch
    permit_sasl_authenticated
    permit_mynetworks
    ...
```

SMTP AUTH gesplittet anbieten

Zwei Instanzen von `smtpd` nutzen unterschiedliche SASL Konfigurationen indem sie unterschiedliche *.conf Dateien laden.

Instanzen und spezifischen Application Name in `master.cf`:

```
# ==========================================================================
# service type              private unpriv  chroot  wakeup  maxproc command + args
#                           (yes)   (yes)   (yes)   (never) (100)
# ==========================================================================
172.16.1.100:smtp      inet  n       -       n       -       -       smtpd
    -o smtpd_sasl_application_name=intern
192.0.34.166:smtp      inet  n       -       n       -       -       smtpd
    -o smtpd_sasl_application_name=extern
```

Clientseitiges SMTP AUTH

Parameter werden global in `main.cf` gesetzt.

SMTP AUTH clientseitig aktivieren:

```
smtp_sasl_auth_enable = yes
```

Verweise auf map mit Authentifizierungsdaten:

```
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
```

Beispiel: `sasl_passwd`

```
# hostname              user:pass
mail.example.com        test:testpass
```

Mechanismen nach Sicherheitsgrad filtern:

```
smtp_sasl_security_options = noanonymous, noplaintext
```

Unerwünschte Mechanismen rausfiltern:

```
smtp_sasl_mechanism_filter = !GSSAPI, static:rest
```

ZUKUNFT VON SMTP AUTH

SMTP AUTH, so wie es heute implementiert ist, wird sehr wahrscheinlich nicht so weiter bestehen.

Nahezu alle Gründe hierfür drehen sich um die Frage: „Wie sicher ist Postfix wenn es Cyrus SASL in seinen eigenen Code mit einbindet und nutzt?“

WIESO MACHT CYRUS SASL POSTFIX UNSICHERER?

Gegenwärtig ist Cyrus SASL code im `smtpd` daemon verankert. `smtpd` ist der „Türsteher“ in der Postfix „Disco“!

Ein kompromittierter `smtpd` ist nur einen `saslauthd` daemon weit von `root` Rechten entfernt.

Cyrus SASL Programmcode soll in einen spezialisierten Postfix daemon wandern. Dieser könnte parallel dazu andere Authentifizierungsdienste wie z.B. `simpleauth` von Peter Bieringer oder GNU SASL unterstützen.

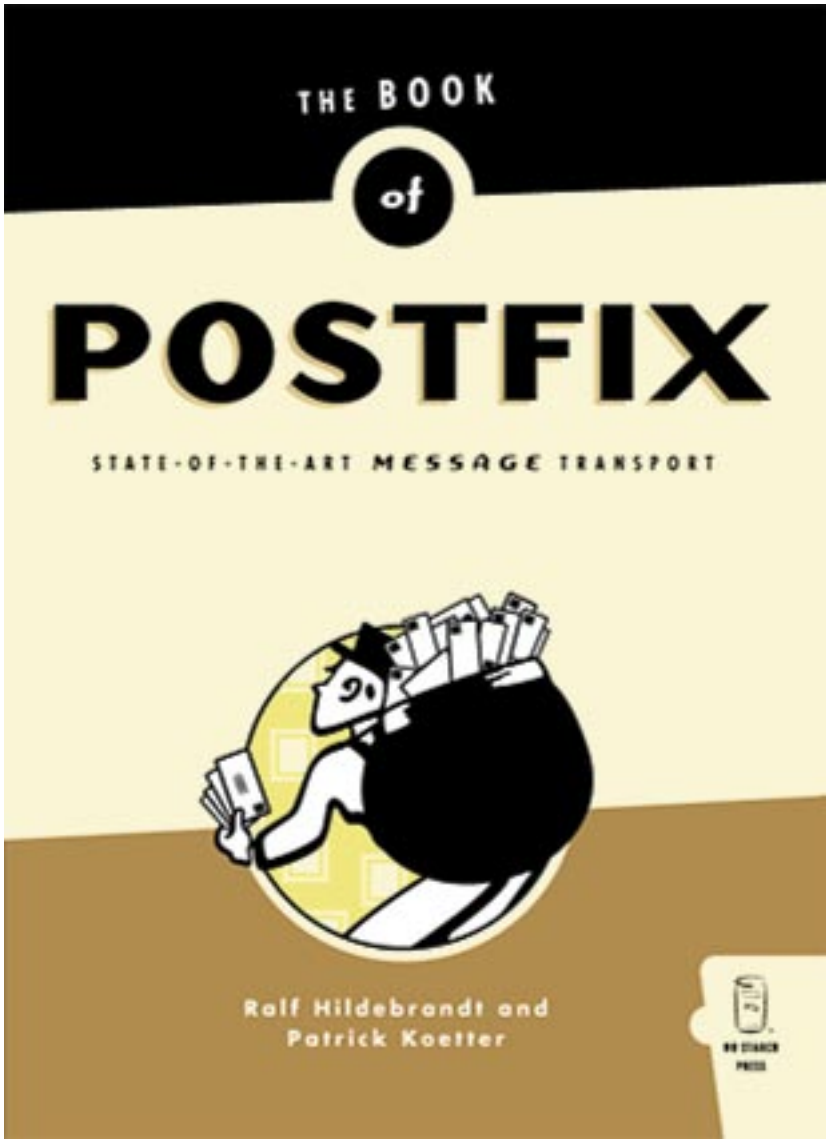
FAZIT: CYRUS SASL

Cyrus SASL mangelt es an brauchbarer Dokumentation und konsistenter, nachvollziehbarer Benamung von Parametern!

Cyrus SASL ist so schlecht dokumentiert, daß man in der Masse von sicherheitkritischen Implementierungsfehlern ausgehen muss.

- Idee ist gut.
- Umsetzung ist komplex.
- Mitgelieferte Dokumentation ist verantwortungslos!

WEITERFÜHRENDE INFORMATIONEN



Kontakt:

Patrick Ben Koetter
Informationsarchitekt
koetter@state-of-mind.de