Simple Authentication and Security Layer

oder

Cyrus SASL
das unbekannte Wesen

Patrick Koetter
1. Postfix Konferenz
4. September 2004, Berlin

EINLEITUNG

Cyrus SASL macht das Konfigurieren von SMTP AUTH für Postfix schwierig.

Grund ist das Fehlen brauchbarer Dokumentation.

Ein Ersttäter muss fast zwangsläufig scheitern!

SIMPLE AUTHENTICATION AND SECURITY LAYER

SASL ist ein Authentication Framework. Es stellt Funktionen zur Verfügung, damit Applikationen, die Authentisierung benötigen, nicht immer „das Rad neu erfinden“ müssen.

Das Framework teilt sich architektonisch in 3 Layer:

Authentication Interface
Mechanism
Method

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

PLAIN

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

LOGIN

berücksichtigt zusätzlich proprietäre Vorgaben für Microsoft Mail Clients

base64 unsicher, denn es läßt sich einfachst decodieren!

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

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

Beispiel für plaintext Authentisierung:

```
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
```

shared-secret

CRAM-MD5

ist der originäre shared-secret Mechanismus.

DIGEST-MD5

ist der Nachfolger.

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 verfügt.

Ist Lösung korrekt, stimmen die Nutzerdaten beider überein.

METHOD

Methods (oder: Password Verification Services) stellen eine Verbindung zwischen den durch Mechanismen übertragenen Daten und Authentication Backends her.

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

Cyrus SASL (Version 2.1.19) verfügt über drei Methods:

– saslauthd (ehem. pwcheck)
- auxprop
- authdaemond

saslauthd

Standalone daemon, der über einen lokalen socket kommuniziert und auf plaintext Mechanismen beschränkt ist.

Verfügbare authentication backends:

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

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

auxprop

Kurzform für auxilliary property plugins, bezeichnet eine Anzahl von authentication backend 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 2.1.19:

- sasldb
- sql

authdaemond

Delegation der Authentifizierung an Courier IMAPs authdaemond daemon.

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 Melnikovs 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
- SASLDB
- SQL

POSIX Accounts

- `/etc/passwd` oder `/etc/shadow` sind bereits vorhanden
- Tools zur Erstellung, Modifikation und Löschung von Accounts bestehen ebenfalls
- SASL muss nichts weiter tun, als einfach darauf zugreifen.

Aber user in diesem authentication backend haben typischerweise SHELL Zugang.

Sensible Daten wie die Authentifizierungsinfos für `root` sind ebenfalls enthalten.

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

PAM

- PAM erweitert Cyrus SASL um neue authentication back-kends
- PAM ist nicht trivial
- PAM ist nicht wirklich hilfreich 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 werden die Benutzerdaten von SASL zu IMAP unverschlüsselt übertragen.

IMAP ist standardmäßig 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, auf Kommunikation über das loopback Interface ausweichen.

LDAP

Sowohl mit saslauthd als auch mit ldapdb möglich.

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

auxprop plugin (noch nicht in 2.1.19) 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 Problemen mit der Berkeley DB ausgesetzt.

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

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

Gibt es andere Zugriffsmethoden, wie z.B. Webinterface?

SQL

Einheitliche Parameter für Lese- und Schreibzugriffe (!) auf folgende SQL server.

- MySQL
- PostGRES
- SQLite

plugin erwartet Kennwörter unverschlüsselt in DB.

Patches, die verschlüsselte Kennwörte in DB möglich machen, sind nicht immer aktuell und evtl. eher ein hack als eine saubere Integration, die sich dauerhaft mit der Cyrus SASL Architektur verträgt.

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 er einen Exploit in Kauf nehmen soll oder schnell mal seine User DB ins Unverschlüsselte migrieren muss.

CYRUS SASL KONFIGURATION

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

Der Name der Konfigurationsdatei setzt sich aus ***Applikations-name*** und Suffix ***conf*** zusammen.

Applikationsname wird vom Server an SASL übermittelt. In Postfix ist **smtpd** der Server. smtpd übermittelt seinen daemon namen an SASL. Der Name der Konfigurationsdatei muss also ***smtpd.conf*** lauten.

Applikationsname seit Postfix 2.1. 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.*`.

Folgende Loglevel existieren:

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
- pwcheck (deprecated)
- auxprop
- authdaemond

`mech_list`

Welche Mechanismen darf der server den clients anbieten?
Gängige Mechanismen:

- PLAIN
- LOGIN
- CRAM-MD5
- DIGEST-MD5
- GSSAPI
- KERBEROS
- OTP

`mech_list` kann nur die Auwahl limitieren! Der client wählt selbstständig einen der angebotenen Mechanismen aus.

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
```

saslauthd

```
# 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.
```

Existiert das Verzeichnis nicht (s.o.), legt man entweder das gewünschte Verzeichnis an - mindestens saslauthd und postfix müssen darauf zugreifen können - oder man lenkt saslauthd mit der Option `m` in ein anderes Verzeichnis:

```
# 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
```

Hier wird deutlich wie uneinheitlich Cyrus SASL Parameter benannt sind. „authentication mechanisms“ heissen seit langem „authentication backends“!

getpwent

getpwent ermöglicht den Zugriff auf das passwd file für System, die kein shadow file 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 das shadow file für System. 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 für `/usr/local/etc/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, z.B.:

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

# auxiliary Plugin parameters
auxprop_plugin: sasldb
```

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

Bei falscher Konfiguration irreführende Fehlermeldungen!

sasldb

Befindet sich sasldb nicht an seinem Standardort oder hat einen anderen Namen, kann man das mit `sasldb_path` konfigurieren:

```
# 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/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 strings:

`%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. Nutzer kann, wenn in OpenLDAP Authorisierung vorliegt, einen `re-bind()` mit den 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
```

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
```

CYRUS SASL TESTEN

Inkonsistenter Parametergebrauch und unterschiedlichste Anforderungen der authentication backends machen es nicht leicht SASL auf das erste Mal gleich „zum Fliegen zu kriegen“.

Größter Fehler:
Funktionsfähigkeit von SASL durch den vorgeschalteten Server, z.B. smtpd, testen. Doppelfehler sind wahrscheinlich und erwartungsgemäß schwierig zu debuggen ist.

SASL zuerst immer ohne den Einfluß irgendeiner anderen Applikation testen; dazu seine eigenen Testtools, `testsaslauthd`, `client` und `server` verwenden.

testsaslauthd

Ein utility um saslauthd basierte Authentifizierung zu testen.

Gelungende Authentifizierung mit `testsaslauthd` kein Beweis, daß die Authentifizierung wirklich klappen wird, denn `testsaslauthd` benützt nicht die libraries von Cyrus SASL!

Sinnvoll ist es daher bevorzugt die utilities server und client zu benutzen!

server

`server` stellt, wie der Name schon nahelegt, die Server-Seite der SASL Test utilities dar. `server` überträgt den ***Applikations-namen*** `sample` an SASL. Konfigurationsdatei muss **sample.conf** heißen.

Wer die Konfiguration eines anderen Servers testen will, spart sich die Tipparbeit und mögliche Fehler und linkt einfach `sample.conf` auf z.b. `smtpd.conf`.

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

`server` unter Angabe eines Services und eines freien Ports starten:

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

Logging funktioniert nicht für alle authentication backends!

client

`client` erwartet, zusätzlich zur Angabe desselben Services und Ports wie `server`, den zu benutzenden Mechanismus und den hostname auf dem sich `server` befindet:

```
# 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
```

SMTP AUTH IN POSTFIX EINRICHTEN

Serverseitiges SMTP AUTH im Schnelldurchflug

```
# /etc/postfix/main.cf
smtpd_sasl_auth_enable = yes
smtpd_sasl_application_name = smtpd
broken_sasl_auth_clients = yes
smtpd_sasl_local_domain =
smtpd_sasl_security_options = noanonymous
smtpd_sasl_exceptions_networks = 172.16.0.0/24
```

Clientseitiges SMTP AUTH im Schnelldurchflug

```
# /etc/postfix/main.cf
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
smtp_sasl_security_options =

# /etc/postfix/sasl_passwd
mail.example.com      test:testpass
```

NEU: smtp_sasl_mechanism_filter

Wahl des zu nutzenden Mechanismus bisher in Postfix client nicht konfigurierbar.

Seit postfix-2.2-20040828 neuer Parameter, `smtp_sasl_mechanism_filter`, der Postfix anweist, alle anderen außer den mit `smtp_sasl_mechanism_filter` gelisteten Mechanismen rauszufiltern:

```
# /etc/postfix/main.cf
smtp_sasl_mechanims_filter = DIGEST-MD5
```

ZUKUNFT VON SMTP AUTH

SMTP AUTH, so wie es heute implementiert ist, wird in Zukunft nicht so weiter bestehen.

Gründe hierfür gibt es viele - nahezu alle drehen sich um die Frage: Wie sicher ist Postfix wenn es Cyrus SASL in seinen eigenen Code mit einbindet und nutzt?

Frage: Wieso sollte Cyrus SASL Postfix unsicherer machen?

Implementierung

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

Wer `smtpd` kompromitiert, kommt sehr wahrscheinlich über Cyrus SASL leichter an das Restsystem, als den Weg über die anderen Postfix daemonen nehmen zu müssen.

Deshalb soll der Cyrus SASL code in einen spezialisierten Postfix Authentisierungs daemon (`sasld`) wandern, der dann parrallel dazu andere Authentifizierungsansätze wie z.B. `simpleauth` von Peter Bieringer oder z.B. GNU SASL unterstützen könnte.

Dokumentation

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

Cyrus SASL ist als authentication framework viel zu bedeutsam geworden, als das man es sich leisten könnte diese Software so schlecht betreut durchs Internet geistern zu lassen.

Die Idee ist gut...
Die Umsetzung komplex...
Die Anleitung verantwortungslos...

VIELEN DANK!

Kontakt:

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