The Exim Mail Transfer Agent

A configuration tutorial

http://www.exim.org

Configuration file

- Exim uses a single runtime configuration file, divided into a number of sections
- The first section contains global option settings
- The other sections start with “begin *sectionname*”
- They are all optional, and may appear in any order
- Comments, macros, if-then-else, and inclusions are available
- Option settings can refer to auxiliary data files, for example, a file of aliases (traditionally **/etc/aliases**)

Configuration file sections

- Global options
  General and input-related options
- Address rewriting rules
  Specify rewriting of envelope and header addresses
- Retry rules
  Control retries after temporary failures
- Router configuration
  Specify recipient address processing
- Transport configuration
  Specify how actual deliveries are done
- Authenticator configuration
  Specify SMTP authentication methods
- Access Control Lists (ACLs)
  Define policy controls for incoming messages

Default configuration file layout

Global option settings

`begin ACL`
Access control lists

required for SMTP input

`begin routers`
Router configuration

`begin transports`
Transport configuration

required for message delivery

`begin retry`
Retry rules

`begin rewrite`
Rewriting rules

`begin authenticators`
Authenticator configuration

Common global options (1)

- SMTP input limits

```
smtp_accept_max = 200
smtp_accept_queue = 150
smtp_accept_reserve = 10
smtp_accept_reserve_hosts = 192.168.0.0/16
smtp_connect_backlog = 100
```

- Overloading

```
queue_only_load = 5
deliver_queue_load_max = 7
```

- Message size limits

```
message_size_limit = 10M
return_size_limit = 65535
```

Common global options (2)

- Spool space check

```
check_spool_space = 100M
check_spool_inodes = 300
```

- Splitting the **input** directory

```
split_spool_directory = true
```

- Parallel remote delivery (per-message, not system wide)

```
remote_max_parallel = 10
```

- Verifying host names

```
host_lookup = 192.168.4.0/24 : \
  192.168.3.0/24
```

- Qualify domain

```
qualify_domain = plc.co.uk
```

Common global options (3)

- Logging

```
log_selector = +incoming_interface \
               +incoming_port \
               +smtp_confirmation \
               -queue_run
```

- Discarding frozen messages

```
ignore_bounce_errors_after = 6h
timeout_frozen_after = 1d
```

- Warning messages

```
delay_warning = 4h:8h:24h
delay_warning_condition = \
  ${if match{$h_precedence:}\
  {(?i)bulk|junk|list}{no}{yes}}
```

Router overview

- Exim contains a number of different routers

  Example: the *dnslookup* router does DNS processing
  the *redirect* router does address redirection
  (aliasing and forwarding)

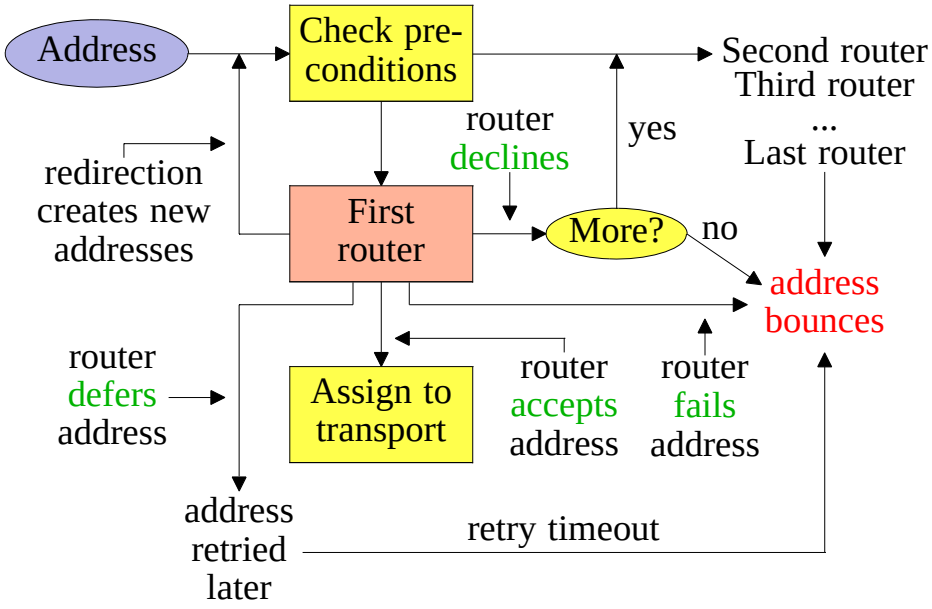
- The configuration defines which routers are used, in which order, and under what conditions

  Example: routers are often restricted to specific domains

- The same router may appear more than once, usually with different configurations

- The order in which routers are defined is important

Exim routing



Simple routing configuration

- Check for non-local domains: run *dnslookup* router
  - Accept: assign to *smtp* transport
  - Decline: “no_more” set, so address bounces
- Check for system aliases: *redirect* router
  - Accept: generates new address(es)
  - Decline: passed to next router
- Check for local user forwarding: another *redirect* router
  - Accept: generates new address(es)
  - Decline: passed to next router
- Check for local user: run *accept* router
  - Accept: assign to *appendfile* transport
- No more routers: address bounces

Exim transports

- Transports are the components of Exim that actually deliver copies of messages

  The *smtp* transport delivers over TCP/IP to a remote host
  The *appendfile* transport writes to a local file
  The *pipe* transport writes to another process via a pipe
  The *lmtp* transport does likewise, using the LMTP protocol
  The *autoreply* transport is anomalous, in that it creates an automatic response instead of doing a real delivery

- The order in which transports are defined is not important
- A transport is used only when referenced from a router
- Transports are run in subprocesses, under their own uid, after all routing has been done

Retry rules

- Fixed or increasing time intervals
- Change of rule as time passes
- Predication on specific errors as well as on host or domain
- Default retry rule:

```
*     *   F,2h,15m;  G,16h,1h,1.5;  F,4d,6h;
```

For all hosts and domains

For all errors

Every 15 mins for 2 hours

Start at 1 hour interval, then increase by 1.5 until 16 hours have passed

Every 6 hours up to 4 days

- Use **-brt** to check retry rules:

```
exim -brt aol.com
Retry rule: aol.com F,2h,15m; F,4d,30m;
```

Temporary delivery errors

- *Host error:* not related to message or recipients
  Host is delayed, for all messages
  No message is sent to it until its retry time has passed
  Retry rule is selected by host or domain

- *Message error:* related to message, but not to recipients
  Message is delayed, to that host only
  Retry rule is selected by host or domain
  Does not affect other messages to that host

- *Recipient error:* specific to one recipient
  Recipient is delayed in all messages, but only in queue runs
  Retry rule is selected by domain or full address

- *Longstop check:* bounce a message that has been on the queue for longer than the host's retry period

Address rewriting (1)

- Global rewriting is done when an address is first seen
  Envelope and header addresses when a message is received
  Addresses generated by aliasing and forwarding

- Rewriting is not intended to be a routing mechanism

- Use with care, for "regularizing" your own addresses

- Use to change host name to corporate domain

  `*@*.plc.co.uk  $local_part@plc.co.uk`

  theboss@hq.plc.co.uk => theboss@plc.co.uk

- Use quotes if pattern or replacement contains white space

- Flag letters control which addresses are rewritten

Address rewriting (2)

- Rewrite login name to real name

```
*@plc.co.uk "${lookup {$local_part}\
             dbm{/etc/realnames}\
             {$value} fail}@$domain" bcfrF
```

theboss@plc.co.uk => J.Caesar@plc.co.uk

- `bcfr` rewrites *bcc*, *cc*, *from*, and *reply-to* header lines
- `F` rewrites the envelope "from" field
- Rewriting rules are applied one by one, in order
  Both the above rules would be applied
- Header addresses and return paths can also be rewritten at transport time

Item lists

- Exim configurations can contain several kinds of list
- Domain list

```
mydomain.example : *.plc.example.com
```

- Host list

```
myhost.example : *.plc.example.com : \
  192.168.3.4 : 192.168.35.0/24
```

- Address list

```
user@dom.com : *@dom.com : user@*.dom.com
```

- Local part list

```
postmaster : \N^abc\d{3,5}$ : \
  ^(?=.*?[a-z])(?=.*?[A-Z])\
   (?=.*?\d).{9}$\N
```

Lookups in item lists

- Item lists are always expanded before being scanned
  The list may be modified by a lookup expansion item
- The lookup facility can also be used as an indexing mechanism
- A plain file name is just an out-of-line list
```
domains = /etc/relaydomains
```
- A true lookup item starts with a lookup type
```
domains = dbm;/etc/relaydomains.db
domains = mysql;select domain from relays \
          where domain = '$domain';
```
- The result of this type of lookup is often not used
- A host list may contain both name and address lookups
```
hosts = dbm;/etc/relaybyname.db :\
        net-dbm;/etc/relaybyip.db :\
        net27-dbm;/etc/relaybyip.db
```
- The last example might lookup "192.168.224.0/27"

Host names in host lists

- Host lists are used to check client hosts
- Initially, all Exim has is the client IP address
  IP address items in host lists can always be checked
- Name items cause DNS lookups, which may fail or time out
- Complete host names cause a forward DNS lookup
```
hosts = mail.exim.org
```
  Look up the address record(s) and then compare IP addresses
- Partial names or lookups cause a reverse DNS lookup
```
hosts = *.exim.org
hosts = cdb;/etc/relay.hosts
```
  Look up the PTR record(s) and then compare names

Negation in item lists

- Lists are scanned from left to right until an item matches
- When items are negated, the order matters a lot

```
domains = a.b.c : !*.b.c : *.c
```

a.b.c matches this list
anything.b.c (except **a.b.c**) does not match this list
anything.c (that is not ***.b.c**) matches this list
Any other domain does not match this list

- If a list ends with a negated item, * is implied at the end

```
hosts = !192.168.45.233
```

192.168.45.233 does not match this list
Any other IP address does match this list

Named item lists

```
domainlist local_domains = @ : plc.com
hostlist   relay_hosts = 192.168.32.0/24
```

- Abstraction: list is specified in one place only
  References are shorter and easier to understand
- Optimization: matches are cached where possible
  Example: several routers testing the same domain list
  Cannot cache by default if list contains expansion items
- A named list is referenced by prefixing its name with +

```
hosts = 127.0.0.1 : +relay_hosts
```

- A named list can be negated

```
domains = !+local_domains
```

This is not possible with macros

Named lists in the default configuration

- The default configuration uses three named lists

```
domainlist local_domains = @
domainlist relay_to_domains =
hostlist   relay_from_hosts = 127.0.0.1
```

- Local domains are going to be delivered on this host
  @ means “the local name of the local host”
- No domains are defined for relaying by default
- The local host is permitted to relay through itself
  Some clients send mail this way
- These lists are used later to define these controls

Default routers (1)

- The first router handles non-local domains

```
dnslookup:
  driver = dnslookup
  domains = ! +local_domains
  ignore_target_hosts = 0.0.0.0 : 127.0.0.0/8
  transport = remote_smtp
  no_more
```

- The precondition checks for a non-local domain
- Silly DNS entries are ignored
- If the domain is found in the DNS, queue for **remote_smtp**
- Otherwise, **no_more** changes “decline” into “fail”

Default routers (2)

- The second router handles system aliases

```
system_aliases:
  driver = redirect
  data = ${lookup{$local_part}lsearch\
         {/etc/aliases}}
  allow_fail                          allows :fail:
  allow_defer                         allows :defer:
  pipe_transport = address_pipe
  file_transport = address_file
# user = exim
```

- Alias files look like this

```
postmaster:  pat, james@otherdom.example
majordomo:   |/usr/bin/majordom ...
alice:       :fail: No longer works here
```

Default routers (3)

- The third router handles users' *.forward* files

```
userforward:
  driver = redirect
  check_local_user
  file = $home/.forward
  no_verify
  no_expn
  check_ancestor
  pipe_transport = address_pipe
  file_transport = address_file
  reply_transport = address_reply
  allow_filter                        allows filter files
```

- **data** and **file** are mutually exclusive options for **redirect**
  data expands to a redirection list
  file expands to the name of a file containing a redirection list

Default routers (4)

- The final router handles local users' mailboxes

```
localuser:
  driver = accept
  check_local_user
  transport = local_delivery
```

- Recap: an address is routed like this:

Remote address	=> **remote_smtp** transport, fail
System alias	=> new address(es), fail, defer
User's *.forward*	=> new address(es)
Local user	=> **local_delivery** transport
Unrouteable address	=> bounce

- This is just one of many possible configurations
  There are other routers that we have not met yet...

Default transports (1)

- Main transports

```
remote_smtp:
  driver = smtp

local_delivery:
  driver = appendfile
  file = /var/mail/$local_part
  delivery_date_add
  envelope_to_add
  return_path_add
# group = mail
# mode = 0660
```

- Default local delivery assumes a "sticky bit" directory
  Setting **group** and **mode** is an alternative approach

Default transports (2)

- Auxiliary transports

```
address_pipe:
  driver = pipe
  return_output

address_file:
  driver = appendfile
  delivery_date_add
  envelope_to_add
  return_path_add

address_reply:
  driver = autoreply
```

Local delivery in maildir format

- Supported by the **appendfile** transport

```
maildir_delivery:
  driver = appendfile
  maildir_format
  directory = /var/mail/$local_part
  ...
```

- Each message is delivered into a separate file
  - A directory rather than a file is specified
  - Messages are written into a subdirectory called *tmp*
  - Once written, they are moved into a subdirectory called *new*
  - The MUA moves a message into *cur* once it has seen it
- MUAs and POP/IMAP servers must support maildir
- Maildir allows multiple simultaneous deliveries and removals
  - No locking is required
- Downside: it is more expensive to calculate disk space usage

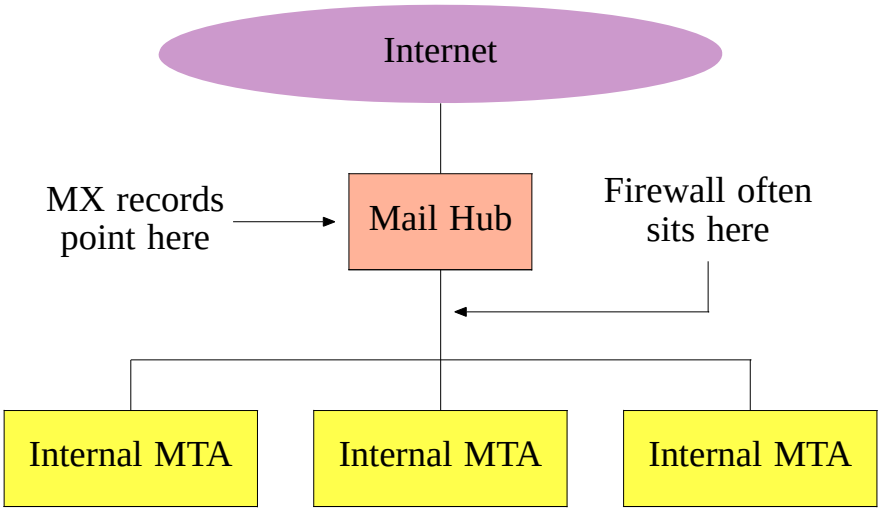
Routing to smarthosts

- Replace the first router with

```
send_to_smarthost:
  driver = manualroute
  domains = ! +local_domains
  route_list = * smarthost1.example:\
                 smarthost2.example
  transport = remote_smtp
```

- A **route_list** rule contains space-separated items
  The first is a single domain pattern: * matches any domain
  The second is a list of hosts for the matching domain
  The third is **bydns** or **byname** (default tries both)
  A transport name may also be given

Mail hubbing



Routing on a mail hub (1)

- For a small number of domains, routes can be inline

```
hubbed_domains:
  driver = manualroute
  route_list = hdom1.plc.ex h1.plc.ex; \
               hdom2.plc.ex h2a.plc.ex:\
                            h2b.plc.ex;\
               *.hq.plc.ex  hqmta.plc.ex
  transport = remote_smtp
```

- Semicolons separate routing rules

- The list of hosts is expanded

```
route_list = *.plc.ex $domain
```

- Router declines if domain fails to match any rule

Routing on a mail hub (2)

- For a large number of domains, use an index from domain to internal host

```
hdom1.plc.ex:  h1.plc.ex
hdom2.plc.ex:  h2a.plc.ex:h2b.plc.ex
...
```

- Use **manualroute** with **route_data**

```
hubbed_domains:
  driver = manualroute
  route_data = ${lookup{$domain}lsearch\
               {/etc/hubbed-domains}}
  transport = remote_smtp
```

- Expansion of **route_data** must yield a list of hosts
  (And optionally **byname/bydns** and/or a transport name)

Other features of **manualroute**

- Can specify "the hosts to which this MX points"

```
route_list = * domain.example/mx
```

- Lists of hosts can be randomized

```
hosts_randomize
route_list = * host1:host2:host3
```

- Randomizing can be in groups

```
hosts_randomize
route_list = * host1:host2:+:host3:host4
```

Group separator

Fallback hosts

- Fallback hosts can be specified on a router

```
dnslookup:
  driver = dnslookup
  fallback_hosts = smarthost.example:...
  ...
```

- Fallback hosts can also be specified on a transport

```
remote_smtp:
  driver = smtp
  fallback_hosts = smarthost.example:...
```

- The transport's fallback hosts are used only if the router did not specify any
- `hosts_randomize` on the transport can be used to randomize fallback hosts

Virtual domains (1)

- Straightforward cases are just an aliasing application

```
virtual_domains:
  driver = redirect
  domains = lsearch;/etc/virtual-domains
  data = ${lookup{$local_part}lsearch\
         {/etc/valias/$domain}}
  no_more
```

- Or use a **dsearch** lookup to save having a separate list

```
  domains = dsearch;/etc/valias
```

  Ensure Exim is built with **dsearch** support

- For large virtual domains, use something better than *lsearch*

Virtual domains (2)

- Add * to the search type to use a default

```
data = ${lookup{$local_part}lsearch*\
       {/etc/valias/$domain}}
```

- Exceptions can be handled with **:fail:**

```
*:          info
freebies:   :fail: Sorry, we ran out
admin:      thedogsbody@domain1.ex
sales:      abc@domain2.ex
info:       xyz@domain3.ex
```

- * is not a wildcard; it just means “default”
  Put it first in **lsearch** files for efficiency

- **no_more** is not needed (but does no harm)

Virtual domains (3)

- Domains can be mixed in a single alias file

```
data = ${lookup{$local_part@$domain}cdb*@\
       {/etc/aliases-mixed.cdb}}
```

- Adding *@ to the search type gives a two-level default
  First default is within the current domain
  Second default is global

- The alias file could look like this

```
*:           lmn@domain6
abc@virt1:   xyz@domain1
*@virt1:     abc@virt1
abc@virt2:   abc@domain2
xyz@virt3:   pqr@domain3
```

Common postmaster for virtual domains (1)

- Common postmaster for all domains

```
postmaster:
  driver = redirect
  local_parts = postmaster
  data = postmaster@your.domain
  repeat_use = false
```

- Put before virtual domains router to handle all domains
  Use a **domains** setting if necessary

- If some virtual domains have their own postmaster
  Put the **postmaster** router after the virtual domains router
  Cannot use **no_more** on virtual domains router
  For domains with defaults, use

```
  postmaster: :unknown:
```

  This forces the router to decline (same as an empty list)

Common postmaster for virtual domains (2)

- The postmaster router must handle unknowns as well when it follows the virtual domains router

```
postmaster:
  driver = redirect
  domains = cdb;/etc/virtual-domains
  allow_fail
  data = ${if eq{$local_part}{postmaster}\
         {postmaster@your.domain}\
         {:fail: Unknown user}}
```

- This is not necessary if all the virtual domains have defaults (But it is a useful safeguard)

Simple mailing lists

- One router can handle many lists

```
lists:
  driver = redirect
  domains = lists.foo.bar
  no_more
  file = /usr/lists/$local_part
  forbid_pipe
  forbid_file
  skip_syntax_errors
  errors_to = $local_part-request@$domain
  syntax_errors_to = \
              $local_part-request@$domain
```

- Error addresses are verified before being used
- Closed lists can also be handled (not shown here)

External local delivery agent

- A transport for **procmail**

```
procmail_pipe:
  driver = pipe
  command = /usr/local/bin/procmail -d \
            $local_part
```

- Router to use **procmail** if the user has a **.procmailrc** file

```
localuser:
  driver = accept
  check_local_user
  transport = ${if exists{$home/.procmailrc}\
    {procmail_pipe}{local_delivery}}
```

- The pipe runs as the local user because of **check_local_user**

Mailboxes without local accounts (1)

- Use a lookup to check for valid local parts

```
no_account_users:
  driver = accept
  local_parts = dbm;/etc/no_accounts.db
  transport = no_account_delivery
```

- The data from the lookup is saved in **$local_part_data**
- For a message store where the files are not individually owned, the transport can be simple

```
no_account_delivery:
  driver = appendfile
  file = /var/mail/$local_part
  user = mail
```

Mailboxes without local accounts (2)

- For individually owned files, keep relevant data with each valid local part (e.g. in **/etc/no_account_db**)

```
user1:  uid=1234 gid=1023
user2:  uid=4567 gid=4242
...
```

- Then the transport can be

```
no_account_delivery:
  driver = appendfile
  file = /var/mail/$local_part
  user = ${extract{uid}{$local_part_data}}
  group = ${extract{gid}{$local_part_data}}
```

- This form of **extract** handles data in *name=value* format

Incoming message control features

- SMTP authentication
- SMTP session encryption using TLS (SSL)
- Local policy is defined in *access control lists* (ACLs)
  - Rules for accepting messages for local delivery
  - Rules for accepting messages for relaying to other hosts
- ACLs can do address verification
  - The delivery routers are used to check envelope addresses
- You can also link into Exim a **local_scan()** function
  - Supports custom checks on incoming messages
  - Written in C to a documented API

Authentication

- SASL – Simple authentication and security layer
  General framework for client-server authentication
  Different authentication “mechanisms”
- Server advertises supported mechanisms
  May be tailored for the client
- Client requests authentication by a specified mechanism
  Data may be included with the request
- Server sends a “challenge” and the client responds
  May be repeated any number of times
- Server accepts or rejects authentication
  235 Successful authentication
  435 Temporary problem with authentication
  535 Authentication failed

Authentication in SMTP

- Mechanisms are advertised in response to EHLO

```
EHLO client.plc.ex
250-server.plc.ex Hello client.plc.ex
250-SIZE 10485760
250-PIPELINING
250-AUTH PLAIN LOGIN
250 HELP
```

- Command is AUTH <*mechanism*> [*data*]
- Challenges use response code 334
- All data is base64 encoded
  Thus, any byte value can be included

PLAIN authentication (RFC 2595)

- Client sends a single set of data, containing three items
  - Identity to login as (not relevant for SMTP)
  - Identity whose password is to be checked
  - The password
- Binary zeros (NULs) separate the three data items
  - `AUTH PLAIN AG15bmFtZQBteXNlY3JldA==`
- Unencoded that is
  - `AUTH PLAIN <nul>myname<nul>mysecret`
- The first field is usually empty in SMTP
- Server responds immediately with success or failure
  - Password is transmitted in cleartext if session not encrypted
  - No challenge is issued; only one exchange is needed
  - (Alternate usage has no data with AUTH, and an empty challenge)

LOGIN authentication

- No formal definition; used by Pine and the c-client library
- Separate challenges (prompts) for username and password

```
AUTH LOGIN
334 VXNlcm5hbWU6                    (Username:)
bXluYW1l                            (myname)
334 UGVzc3dvcmQ6                    (Password:)
bXlzZWNyZXQ=                        (mysecret)
235 Authentication successful
```

- The password is again passed in cleartext
  - Three exchanges are required
- Some clients are picky about the exact text of the prompts

CRAM-MD5 authentication (RFC 2195)

- Server sends a challenge string that is different each time

```
AUTH CRAM-MD5
334 PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2Uuc...
 (<1896.697170952@postoffice.reston.mci.net>)
```

- Client sends back a username, and the MD5 digest of the challenge string concatenated with the password (in hex)

```
dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZ1NzMzNG...
 (tim b913a602c7eda7a495b4e6e7334d3890)
```

- Server does the same computation, compares the result, and accepts or rejects

```
235 authentication successful
```

- The password does not traverse the network
  But it must be stored in cleartext at both ends

SMTP authentication in Exim

- Different authenticator drivers for different mechanisms
  Can be configured for server or client or both

- On an Exim server
  AUTH is advertised if the client matches **auth_advertise_hosts**
  This is expanded, so can depend on circumstances
  For example, it can be empty unless the session is encrypted

- On an Exim client, authentication is attempted if
  The server is in **hosts_require_auth** or **hosts_try_auth**
  (options of the **smtp** transport), and ...
  A client authenticator matches an advertised mechanism

- On failure, Exim delivers unauthenticated for **hosts_try_auth**

The **plaintext** authenticator

```
plain:
  driver = plaintext
  public_name = PLAIN
  server_prompts = :
  server_condition = ${if and {{eq{$2}
    {myname}}{eq{$3}{mysecret}}}{yes}{no}}
  server_set_id = $2
  client_send = ^myname^mysecret

login:
  driver = plaintext
  public_name = LOGIN
  server_prompts = Username:: : Password::
  server_condition = ${if crypteq{$2}\
   {${lookup{$1}lsearch{/etc/passwd}\
   {${extract{1}{:}{$value}}}fail}}{yes}{no}}
  server_set_id = $1
  client_send = : myname : mysecret
```

The **cram_md5** authenticator

```
cram:
  driver = cram_md5
  public_name = CRAM-MD5
  server_secret = ${lookup{$1}dbm\
    {/cram/secrets}{$value}fail}
  server_set_id = $1
  client_name = tim
  client_secret = tanstaafl
```

- It is important to specify **fail** for a failing lookup
- Why is this version wrong?

```
server_secret = ${lookup{$1}dbm\
  {/cram/secrets}}
```

- Answer: it provides an empty secret for unknown users

Encrypted SMTP connections

- TLS (transport layer security) aka SSL (secure socket layer)
  Exim uses the OpenSSL or GnuTLS library for TLS support
- Server advertises support for the STARTTLS command
  Client issues STARTTLS
  Server gives positive response
  An encryption key is then negotiated according to TLS rules
  Subsequent data is encrypted before transmission
  Session state is reset; a new EHLO is needed
- Exim can be configured to support the obsolete **smtps** protocol
  But not both at once, since TLS is assumed for all connections
- Messages are not encrypted while in the hosts at either end
  TLS gives protection only against eavesdroppers
  In particular, it provides protection for AUTH passwords
- Client certificates can (alternatively) be used for authentication

TLS on an Exim server

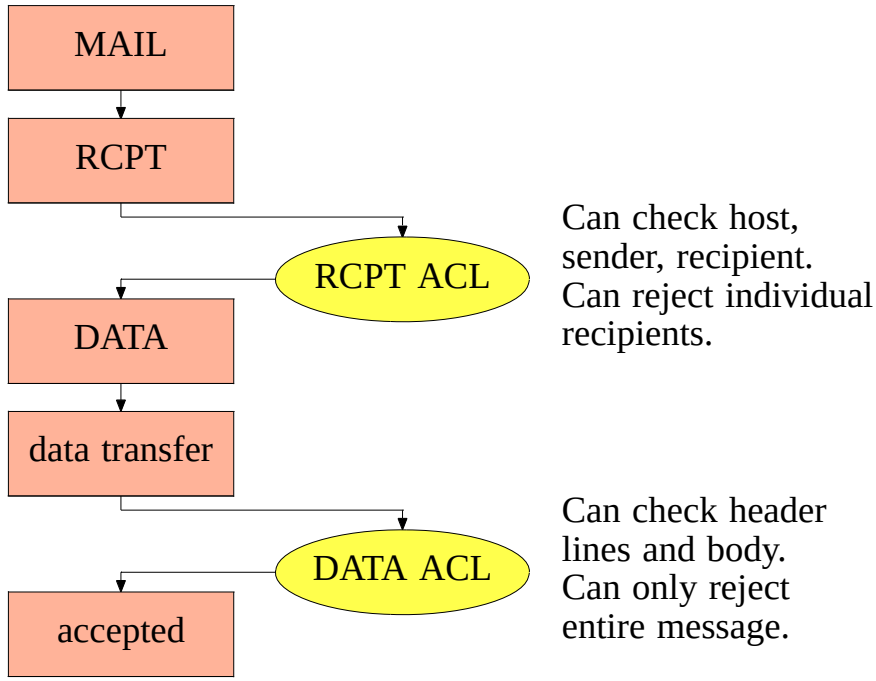
- Three options must be set for TLS to be used at all
  tls_certificate
    the file containing the server's certificate
  tls_privatekey
    the file containing the server's private key
  tls_advertise_hosts
    specifies which clients should be told
- The *exim* user must be able to read the private key
- To verify client certificates
  tls_verify_certificates
    the file containing the expected certificates
  tls_verify_hosts
    specifies clients that must be verified
  tls_try_verify_hosts
    specifies clients that may be verified

TLS on an Exim client

- Exim will try to use TLS by default if the server advertises it
  ...if Exim is built with TLS support!
- The following options are set on the **smtp** transport
  Expansion allows for different values for different servers
- Set **tls_certificate** and **tls_privatekey** for client certificate
  Used only if the server requests a certificate
- Set **hosts_avoid_tls** to suppress encryption for specific servers
- Set **hosts_require_tls** to insist on encryption
  Otherwise, Exim will send in clear if STARTTLS is rejected
- Set **tls_verify_certificates** to verify the server's certificate
- Set **tls_require_ciphers** to restrict which ciphers are used

Access control lists

- Most ACLs are relevant for SMTP input
  They do apply to local (*stdin/stdout*) SMTP (the **-bs** option)
  An ACL is available for non-SMTP input
- For incoming SMTP messages the main ACLs are these
  acl_smtp_rcpt defines the ACL to be run for each RCPT
    Default is "deny"
  acl_smtp_data defines the ACL to be run after DATA
    Default is "accept"
- Tests on message content can be done only after DATA or in a non-SMTP ACL
- Other ACLs can be user for AUTH, ETRN, EXPN, EHLO, MAIL, STARTTLS, VRFY, and at start of an SMTP session



A simple ACL

- In the main section of the configuration

```
acl_smtp_rcpt = acl_check_rcpt
```

- In the ACL section of the configuration

```
acl_check_rcpt:
  accept   local_parts = postmaster
           domains     = +my_domains

  require  verify      = sender

  accept   domains     = +my_domains
           verify      = recipient
```

- Conditions are “anded” together
  - Conditions may be repeated
  - Evaluation is in order
  - Evaluation stops as soon as the outcome is known
- Implicit “deny” at the end

Finding an ACL

- **acl_smtp_rcpt** etc. are expanded, and can then be:
  An absolute file name (the file contains the ACL)
  The name of an ACL in the configuration (previous example)
  The text of an ACL itself

```
acl_smtp_vrfy = accept
```

- Choice of ACL can be made to depend on client host, sender address, recipient address, day of the week, or anything else that Exim knows about

```
acl_smtp_rcpt = ${if eq \
  ${mask:$sender_host_address/24}\
  {10.1.2.0/24}\
  {acl_local}{acl_remote}}
```

ACL statements

- Each statement contains a verb and a list of conditions

verb	*condition 1* (one per line)
	condition 2
	...

- If all the conditions are satisfied

accept	Accepts SMTP command or non-SMTP message (else may pass or reject – see later)
defer	Gives a temporary rejection (= **deny** for non-SMTP)
deny	Rejects (else passes)
discard	Like **accept** but discards recipients
drop	Like **deny** but drops an SMTP connection
require	Passes (else rejects)
warn	Takes some warning action (writes log or adds header) Always passes

ACL conditions and modifiers

- When a condition is false, no subsequent ones are evaluated
- Modifiers can appear among the conditions

  message is used when access is denied

```
require  message = sender must verify
         verify  = sender
         message = recipient must verify
         verify  = recipient
```

- Modifiers that follow a false condition are not processed

  This example does not work

```
require  verify  = sender
         message = sender must verify
```

ACL modifiers (1)

- **message** defines a custom message for a denial or warning

```
deny    message  = You are black listed at \
                   $dnslist_domain
        dnslists = rbl.mail-abuse.org : ...
```

- **log_message** defines a custom log message

```
require log_message = Recipient verify failed
        verify      = recipient
```

- **endpass** is used with **accept** for a 3-way outcome

```
accept  domains = +local_domains
        endpass
        verify  = recipient
```

  Above **endpass**, failure causes the next statement to be run

  Below **endpass**, failure causes rejection

ACL modifiers (2)

- **control** can specify message freezing or queueing

```
accept hosts   = ...
       control = queue_only
```

- **delay** causes Exim to wait before continuing

```
deny  !verify = recipient
       delay = 60s
```

- **set** sets ACL variables

```
warn   condition = ...
       set acl_m4 = value
```

acl_m*x* variables remain set for the message
acl_c*x* variables remain set for the connection

- Values when message is accepted are available during delivery

The default ACL (1)

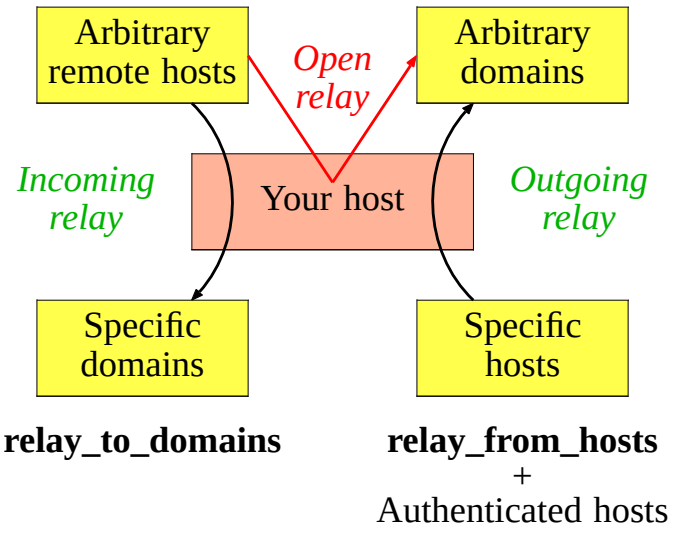
```
acl_check_rcpt:

  accept  hosts         = :

  deny    domains       = +local_domains
          local_parts   = ^[.] : ^.*[@%!/|]

  deny    domains       = !+local_domains
          local_parts   = ^[./|] : \
                          ^.*[@%!] : \
                          ^.*/\\.\\./

  accept  local_parts   = postmaster
          domains       = +local_domains

  require verify        = sender
```

(continued)

The default ACL (2)

```
accept  domains       = +local_domains
        endpass
        message       = unknown user
        verify        = recipient

accept  domains       = +relay_to_domains
        endpass
        message       = unrouteable address
        verify        = recipient

accept  hosts         = +relay_from_hosts

accept  authenticated = *

deny    message       = relay not permitted
```

Good and bad relaying



DNS black lists (1)

- Default is to look up the client host's IP address

```
deny  message = rejected because $sender_\
        host_address is in a black list \
        at $dnslist_domain\n$dnslist_text
      dnslists = sbl.spamhaus.org : ...

warn  message = X-Warning: \
        $sender_host_address is in a black \
        list at $dnslist_domain
      log_message = found in $dnslist_domain
      dnslists = dialups.mail-abuse.org
```

- You can also look up mail domains

```
deny  message = sender's domain is listed \
        at $dnslist_domain
      dnslists = dsn.rfc-ignorant.org/\
                 $sender_address_domain
```

DNS black lists (2)

- The RHS value can be specified

```
deny dnslists = \
       rblplus.mail-abuse.org=127.0.0.2
```

- The value is in **$dnslist_value** during message expansion
- DNS list lookups are cached for each incoming message
  (Not repeated for each recipient)

Verifying addresses

- Verification asks: *Could we deliver to this address?*
- Check by running the address through the routers
- "Verify mode" is set: routers can behave differently
  Skip this router if **no_verify** is set
  Use this router only for verification if **verify_only** is set
  Fail instead of accept if verifying and **fail_verify** is set
- The verification conditions for envelope addresses are

```
verify = sender
verify = recipient
```

- Verification defers can be allowed to pass

```
require verify = sender/defer_ok
```

Verification "callouts"

- Routers can check only the domains of remote addresses
- Callouts can be used to do more

```
require verify = sender/callout
```

- Connects to the routed host and checks with a RCPT command
  This is expensive but a cache is used
  Callouts can be used with recipients as well as senders
  "Random" and postmaster checks can be requested
- Callout defers can be allowed to pass

```
require verify = recipient/callout=defer_ok
```

- Callouts can only answer "no" or "maybe"
- Callouts do not stop much spam nowadays
  Most spam messages have a valid (forged) sender

Verifying header syntax

```
require verify = header_syntax
```

- Checks those header lines that contain addresses
  From: To: Cc: Bcc: Reply-To: Sender:
- Can be used only after DATA or in the non-SMTP ACL
- Catches syntactic junk

```
To: @
To: Undisclosed recipients
To: abc@x.y.z <abc@x.y.z>
To: <>
```

- Rejects unqualified addresses by default
  Set **sender_unqualified_hosts** or **recipient_unqualified_hosts**

Verifying a header sender address

```
require verify = header_sender[/options]
```

- Ensures that there is a valid sender in at least one header line
  Checks **Sender:**, **Reply-To:**, and **From:**
- Can be restricted to bounce messages only

```
deny senders = :
     message = Need valid header sender
    !verify  = header_sender
```

- **senders** checks the envelope sender address
  Empty item checks for empty sender (bounce message)

Requiring encryption

- Can check for specific ciphers

```
deny message = wrong cipher
     encrypted = DES-CBC3-SHA
```

- Use * to check for any cipher (i.e. check for any encryption)
- Can check a client's certificate

```
accept verify = certificate
```

Requires **tls_verify_hosts** or **tls_try_verify_hosts** to be set

- Insisting on encryption for authentication

```
auth_advertise_hosts = ${if \
    (main option)          eq{$tls_cipher}{}\
                           {}{*}}
accept encrypted = *       (in ACL for AUTH)
```

More complex ACL for AUTH

- Suppose you want to allow all authentication mechanisms on encrypted connections, but only CRAM-MD5 when the session is not encrypted
- Use this ACL to control the AUTH command

```
acl_check_auth:
  accept encrypted = *
  accept condition = ${if eq \
           {${uc:$smtp_command_argument}}\
           {CRAM-MD5}{yes}{no}}
  deny   message = Require CRAM-MD5 or \
                   TLS encrypted connection
```

- **$smtp_command_argument** is set in non-message ACLs

The Exiscan patch

- Exiscan is a patch that is maintained by Tom Kistner
- It adds conditions to the DATA ACL

  demime sanity checks on MIME structure
  also does extension checking

  malware detects viruses and other malware using 3rd party
  scanners such as Sophos

  spam uses results from SpamAssassin

  regex does regex matches on a message
- Each condition passes back expansion variables that contain useful information
- Get Exiscan from **http://duncanthrax.net/exiscan-acl/**

Nested ACLs

- Calling a nested ACL

```
accept verify = recipient
       acl    = ${lookup{$local_part}dbm\
         {/etc/per-user/acls}{$value}fail}
```

- Forced **fail** in a condition expansion ignores the condition
  The above example accepts if the lookup forces failure
- An empty ACL causes the condition to fail
  Without **fail**, the above example denies if the lookup fails

The **local_scan()** function

- An installation can supply its own **local_scan()** function
  Written in C and linked into the Exim binary
- Called just before a message is accepted, after all other tests
- Can inspect header lines (in main memory) and body (on disk)
- Can reject the message with a custom error message
  Permanent or temporary rejection
- Can accept the message
  Add or remove header lines
  Modify the recipients list (no recipients means “discard”)
  Supply a string for **$local_scan_data**

Testing policy controls

- The **-bh** option runs a fake SMTP session

```
exim -bh 192.203.178.4
>>> host in host_lookup? yes (matched "*")
>>> looking up host name for 192.203.178.4
>>> IP address lookup yielded dul.crynwr.com
>>> checking addresses for dul.crynwr.com
>>>   192.203.178.4
>>> host in host_reject_connection? no
                                (option unset)
...
LOG: SMTP connection from dul.crynwr.com
                               [192.203.178.4]
220 your.host.name ESMTP Exim 4.32 Sun,
                                     10 Apr...
<enter SMTP commands here>
```

Exim is available from

ftp://ftp.csx.cam.ac.uk/pub/software/email/exim/...

.../exim4/exim-4.*xx***.tar.gz** (or **.bz2**) is the latest release

- GNU General Public Licence
- ASCII documentation included
- PostScript, PDF, Texinfo, and HTML are also available
- FAQ in ASCII and HTML with keyword-in-context index
- See also: **http://www.exim.org**
- Discussion list: **exim-users@exim.org**
- Announce list: **exim-announce@exim.org**
- Indexed archive: **http://www.exim-users.org**