postfwd

Das flexible Policy-Framework

Jan Peter Kessler
IT Consulting & Security
info@postfwd.org
http://www.postfwd.org

4. Mailserver-Konferenz
Berlin, 02. Juli 2009

Informationen zur Person

- Seit etwa 12 Jahren im Bereich IT-Sicherheit im RZ-Umfeld tätig; Freiberufler seit 2004

- Schwerpunkte:

  UNIX, Firewalls, Intrusion Detection/Prevention, Proxysysteme, Content-Filter, VPNs und klassische DMZ Dienste (DNS, Web, Mail)

Warum Policy Server (1/3)?

- Content-Filter wie SpamAssassin benötigen erhebliche Ressourcen. Ein Großteil des ‚handelsüblichen' Spams lässt sich jedoch bereits an Übermittlungsdaten (während der SMTP Transaktion) erkennen.

- Bestimmte Prüfungen sind mit Postfix Bordmitteln kaum oder gar nicht möglich (RBL basiertes White- oder Greylisting, zeitliche Steuerung, Rate Limits)

- Einzelne Kriterien, wie z.B das Listing auf einer RBL sind dabei gelegentlich nicht ausreichend zuverlässig

Warum Policy Server (2/3)?

- Konfigurationen mit kombinierten Kriterien durch Postfix Restriction Classes werden in großen, mandantenfähigen Umgebungen schnell unübersichtlich, Beispiel:

```
/etc/postfix/main.cf:
smtpd_restriction_classes = restrictive, permissive
restrictive = reject_unknown_sender_domain reject_unknown_client ...
permissive = permit
smtpd_recipient_restrictions =
        permit_mynetworks
        reject_unauth_destination
        check_recipient_access hash:/etc/postfix/recipient_access

 /etc/postfix/recipient_access:
joe@my.domain permissive
jane@my.domain restrictive
```

Warum Policy Server (3/3)?

- Lösung: Access Policy Delegation

  Postfix übermittelt dabei eine Reihe verschiedener Attribute zur SMTP-Transaktion an ein externes Programm und tritt somit die Zugriffs-entscheidung an dieses ab. Beispiel:

```
request=smtpd_access_policy
protocol_state=RCPT
protocol_name=SMTP
helo_name=some.domain.tld
queue_id=8045F2AB23
sender=foo@bar.tld
recipient=bar@foo.tld
client_address=1.2.3.4
client_name=another.domain.tld
instance=123.456.7
sasl_method=plain
sasl_username=you
sasl_sender=
size=12345
[leere Zeile]
```

Warum postfwd (1/2)?

- Flexible Zugriffssteuerung durch freie Kombination aller Elemente des Policy Delegation Protokolls in einem Firewall ähnlichen Regelwerk ohne Programmierkenntnisse
- Zusätzliche Prüfungen wie Datum/Zeit, Sender DNS (NS, MX)
- Asynchrone Abfrage von DNSBLs mit frei definierbaren Aktionen und automatischer Deaktivierung bei Timeouts
- Makros und Sprünge im Regelwerk

Warum postfwd (2/2)?

- Punkte-System (Scoring) auf Basis freier Kriterien
- Abfragen anderer Policy Services (z.B. postgrey)
- Rate Limits basierend auf Anzahl von Übertragungsversuchen oder Nachrichtengröße
- Interner Cache für komplette Zugriffsversuche oder DNS Abfragen
- Internes Statistiksystem zur Effizienzanalyse einzelner Regeln
- Ausführliche Dokumentation
- Open Source (BSD Lizenz)

Was leistet postfwd NICHT?

- Keine Default Konfiguration

  Zur sinnvollen Nutzung von postfwd muss ein individuelles Regelwerk erstellt werden

- Kein Content-Filter

  Policy Server haben prinzipbedingt keinen Zugriff auf Nachrichteninhalte

Einsatzbeispiele

- Mehrere DNSBLs auswerten:

```
rbl=zen.spamhaus.org,ix.dnsbl.manitu.net,bl.spamcop.net;
rblcount=2;  action=REJECT listed on $$dnsbltext
```

- Nachts sind alle Katzen grau:

```
id=GREY01; client_name==unknown; action=greylisting
id=GREY02; time!=06:00-22:00; action=greylisting
```

- Kriterien kombinieren:

```
id=R01; action=score(-0.4);
client_name!=unknown;  rbl=list.dnswl.org

id=R02; action=score(0.5);
client_name==unknown;  sender_mx_addrs=10.0.0.0/8, 172.16.0.0/12
```

Voraussetzungen zum Einsatz

- Postfix Version 2.1 oder höher
- Perl 5.6.1, empfohlen Perl 5.8 und höher
- Perl Module Net::DNS und Net::Server
  Installation direkt von http://search.cpan.org oder
  via Kommandozeile:

```
# perl -MCPAN -e shell
cpan shell -- CPAN exploration and modules installation (v1.7602)

cpan> install Net::DNS Net::Server
```

Erstellen eines Regelwerkes (1/4)

- Ohne definiertes Regelwerk gibt postfwd „DUNNO“ (keine Restriktion gefunden) an Postfix zurück

- Regeln können in einer Datei (--file=/some/where) hinterlegt oder direkt via Kommandozeile (--rule=“…“) übergeben werden

```
# postfwd –f /some/where
```

- Mehrere Regelsätze können durch multiplen Aufruf übergeben werden. Die Reihenfolge bleibt dabei erhalten:

```
# postfwd –rule=“…“ –file=/some/where/1 –file=/some/where/2
```

- Regelwerke werden top-down von postfwd verarbeitet

Erstellen eines Regelwerkes (2/4)

- Zur Erstellung einzelner Regeln gilt folgende Syntax:

  ```
  <item1> = <value1> ; <item2>=<value2>; action=<action>
  ```

  also:

  wenn item1=value1 UND item2=value2 DANN return action

  Beispiel:

  ```
  client_address=1.1.1.1; sender_domain=domain.tld; action=REJECT
  ```

- Natürlich können dabei beliebig viele Elemente zu einer Regel kombiniert werden.

Erstellen eines Regelwerkes (3/4)

- Zur Identifikation von Regeln im Log und den Statistiken, sollte eine eindeutige ID zu jeder Regel vergeben werden:

```
id=RULE01; client_address=1.1.1.1; action=REJECT
```

Logeintrag bei Eintreffen eines passenden Requests:

```
Jun  1 12:08:02 mail postfwd[164]: [RULES] rule=62, id=RULE01,
client=unknown[1.1.1.1], sender=<jane@doe.local>,
recipient=<info@postfwd.org>, helo=<some.com>, proto=SMTP,
state=RCPT, delay=1s, hits=RULE01, action=REJECT
```

- Ist keine ID definiert, vergibt postfwd selbst eine eindeutige

Erstellen eines Regelwerkes (4/4)

- Kommentare werden mit einem „#“, Regeln über mehrere Zeilen durch einen abschließenden „\“ gekennzeichnet:

```
# this client has misbehaved several times
id=RULE01; action=REJECT due to abuse \
     client_address=1.1.1.1
```

- Die Reihenfolge einzelner Elemente spielt keine Rolle. Whitespaces werden ignoriert.

```
# this is equal…
client_address=1.1.1.1; action=REJECT

# …to this
action=REJECT ; client_address = 1.1.1.1
```

- Nicht interpretierbare Regeln werden ignoriert. Findet postfwd gar keine Regeln vor, wird „DUNNO“ an Postfix übermittelt und eine entsprechende Warnung im Log vermerkt.

Testen des Regelwerkes (1/2)

- Mittels „--showconfig“ Option die Interpretation der Regeln anschauen:

```
# postfwd --showconfig --rule='client_address=1.1.1.1; action=REJECT'
Rule   0: id->"R-0"; action->"REJECT"; client_address->"=;1.1.1.1"
```

- Mittels „--nodaemon“ einen Request auf der Kommandozeile testen. „-L“ leitet die Logs auf die Kommandozeile um:

```
# cat request.sample | postfwd -L --nodaemon -f /etc/postfwd.cf
action=REJECT

[LOG info]: [RULES] rule=62, id=RULE01, client=unknown[1.1.1.1],
sender=<jane@doe.local>, recipient=<info@postfwd.org>, helo=<some.com>,
proto=SMTP, state=RCPT, delay=1s, hits=RULE01, action=REJECT
```

Testen des Regelwerkes (2/2)

- Loglevel erhöhen:

  Die Optionen –v und –vv zeigen detaillierte Informationen zur Verarbeitung. (Vorsicht letztere ist wirklich sehr gesprächig!). Postfwd2 führt außerdem Debug Klassen ein, mit denen gezielt bestimmte Vorgänge untersucht werden können.

```
[LOG info]: [RULES] rule: 0, id: R-0, items: 'action;id;client_name'
[LOG info]: compare client_name:  "english-breakfast.cloud9.net"  "=~"  "\.google\.de$"
[LOG info]: type default :  "english-breakfast.cloud9.net"  "=~"  "\.google\.de$"
[LOG info]: match client_name:  FALSE
[LOG info]: count client_name:  request=0  minimum: 1  result: FALSE
[LOG info]: compare client_name:  "english-breakfast.cloud9.net"  "=~"  "\.web\.de$"
[LOG info]: type default :  "english-breakfast.cloud9.net"  "=~"  "\.web\.de$"
[LOG info]: match client_name:  FALSE
[LOG info]: count client_name:  request=0  minimum: 1  result: FALSE
[LOG info]: [RULES]  RULE: 0  MATCHES: 0
```

- Der ultimative Test ist immer „live“:

  Die Option „-t|--test“ lässt postfwd alle Regeln normal verarbeiten, gibt aber immer „DUNNO“ an postfix zurück.

Verarbeitung des Regelwerks (1/3)

- Der Parser prüft die Elemente eines Policy Delegation Requests gegen das postfwd Regelwerk und löst ggf die konfigurierte Aktion (action=) aus. Ähnlich einer klassischen Firewall gilt eine Regel als wahr, wenn jedes Element des Regelwerkes (bzw eines aus jeder Elementliste) beim Vergleich zutrifft. D.h. folgende Regel:

  ```
  client_address=1.1.1.1, 1.1.1.2; client_name=unknown; action=REJECT
  ```

  löst einen REJECT aus, wenn die Client Adresse (1.1.1.1 ODER 1.1.1.2) UND der Client Name „unknown“ ist.

- Mehr Informationen dazu unter
  http://www.postfwd.org/doc.html#parser

Verarbeitung des Regelwerks (2/3)

- Bei mehrfachem Vorkommen eines Elementes wird eine Elementliste gebildet:

```
# postfwd1 -Cv
      --rule='action=dunno;
        client_name~=/\.google\.de$/;
        client_name~=/\.web\.de$/'
```

ergibt

```
==============================================================
Rule count: 1
==============================================================
Rule   0: id->"R-0"; action->"dunno"
       client_name->"=~;\.google\.de$, =~;\.web\.de$"
--------------------------------------------------------------
```

- Adress- und RBL-Elemente können CSV-separiert werden:

```
rbl=zen.spamhaus.org, ix.dnsbl.manitu.net, bl.spamcop.net
client_address=192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8, 127.0.0.1/8
```

Verarbeitung des Regelwerks (3/3)

- Der Vergleich von Elementen kann durch den Operator beeinflusst werden:

```
=====================================================
 ITEM==VALUE                    true if ITEM equals VALUE
 ITEM=>VALUE                    true if ITEM >= VALUE
 ITEM=<VALUE                    true if ITEM <= VALUE
 ITEM=~VALUE                    true if ITEM ~= /^VALUE$/i
 ITEM!=VALUE                    false if ITEM equals VALUE
 ITEM!>VALUE                    false if ITEM >= VALUE
 ITEM!<VALUE                    false if ITEM <= VALUE
 ITEM!~VALUE                    false if ITEM ~= /^VALUE$/i
 ITEM=VALUE                     default behaviour
=====================================================
```

- Informationen zum sog. „default behaviour“ finden sich unter http://www.postfwd.org/doc.html#items. I.d.R. gilt dabei: Namen => PCRE, Werte => Numeric und Adressen => CIDR

Elemente eines Regelwerkes (1/5)

- Jedes Element, dass Postfix übermittelt, kann im Regelwerk geprüft werden (auch künftige Erweiterungen).

- Ist ein Element im Regelwerk, dass Postfix nicht übermittelt (z.B. ccert_fingerprint unter Postfix < 2.1), wird die Regel ignoriert.

- Eine vollständige Liste aller gültigen Elemente findet sich in der Postfix Dokumentation unter http://www.postfix.org/SMTPD_POLICY_README.html#protocol

Elemente eines Regelwerkes (2/5)

- Neben den dort genannten Elementen bietet postfwd weitere
  http://www.postfwd.org/doc.html#items

- Beispiele

  `rbl=<liste>, rhsbl=<liste>`

  Ermöglicht das parallele Abfragen von DNSBLs.

  Bsp: `rbl=zen.spamhaus.org, ix.dnsbl.manitu.net, bl.spamcop.net`

  `rblcount=<cnt>, rhsblcount=<cnt>`

  Gibt an, dass mindestens <cnt> Listings vorhanden sein müssen, damit das Element wahr wird.

  Bsp: `rblcount=2;rbl=zen.spamhaus.org,ix.dnsbl.manitu.net,bl.spamcop.net`

Elemente eines Regelwerkes (3/5)

`date=<datum>, time=<zeit>`

`days=<days>, months=<months>`

Ermöglicht zeitgesteuerte Verarbeitung.

Bsp: `days=Tue-Fri; months=Apr-Jun; date=24.12.2009-26.12.2009`

`sender_domain=<domain>, recipient_domain=<domain>`

Direkter Zugriff auf den Domainpart.

Bsp: `sender_domain==domain.tld`

`sender_ns_names=<name>, sender_ns_addrs=<addr>`

`sender_mx_names=<name>, sender_mx_addrs=<addr>`

Zugriff auf DNS Informationen zur Senderdomain.

Bsp: `sender_mx_address=192.168.0.0/16`

Elemente eines Regelwerkes (4/5)

- Außer statischen Werten können auch andere Elemente eines Requests zum Vergleich mittels des „$$“-Präfix‘ genutzt werden:

```
id=R001; sender==$$recipient
```

- Sich wiederholende Listen können mittels „&&“-Präfix zu Makros zusammengefasst werden:

```
&&RBLS = { rbl=zen.spamhaus.org, bl.spamcop.net; };
id=R001; &&RBLS; client_name==unknown; action=REJECT
id=R002; &&RBLS; helo_name~=/^[^\.]+$/; action=REJECT
```

Elemente eines Regelwerkes (5/5)

- Seit postfwd1 v1.15RC1 und postfwd2 v0.18RC1 können Listen in separate Dateien ausgelagert werden (1 Element pro Zeile):

```
id=R001; rbl=file:/etc/postfwd/rbls; action=REJECT
```

```
/etc/postfwd/rbls:
zen.spamhaus.org
bl.spamcop.net
# Dateien duerfen andere Dateien und Kommentare enthalten
file:/etc/postfwd/rbls2
```

- Diese Dateien werden einmalig während der Konfigurationsphase geladen. Soll der Inhalt bei jedem Request auf Veränderungen geprüft werden, ist lfile: zu verwenden:

```
id=R001; rbl=lfile:/etc/postfwd/rbls; action=REJECT
```

Aktionen (1/4)

- Es kann jede gültige Aktion an Postfix zurückgegeben werden. Neben den üblichen OK, DUNNO und REJECT auch weitere Restriktionen (z.B. reject_unknown_client) oder Restriction Classes.

- Nach der Rückgabe einer solchen Aktion wird die weitere Verarbeitung abgebrochen.

- Des weiteren kennt postfwd eigene Aktionen, die ggf eine weitere Verarbeitung ermöglichen.

Aktionen (2/4)

- Beispiele:

`jump(<id>)`

Springt zur Regel mit der ID <id>. Ignoriert ungültige Ids.

```
id=R001; size>=100000; action=jump(DEFAULT)
```

`note(<text>)`

Vermerkt <text> im Protokoll (syslog).

```
id=R001; size>=100000; action=note($$client_address sent too big)
```

`score(<score>)`

Verändert den Score eines Requests um den angegebenen Wert (+ - * / und =).

```
id=R001; size>=100000; action=score(+3.2)
```

Aktionen (3/4)

`set(<var>=<value>)`

Setzt eine frei Variable mit einem Wert zur späteren Verarbeitung (+ - * / und =).

```
id=R01; rbl=zen.spamhaus.org, ... ; action=set(CNT=$$rblcount)
id=R02; CNT>= 2; action=REJECT listed on $$CNT rbls
```

`rate(<item>/<max>/<time>/<action>)`

Setzt ein Rate Limit von <max> Requests innerhalb <time> Sekunden für <item>.

Bsp:
```
id=R001; size>=100000; \
    action=rate($$client_address/3/60/REJECT)
```

`ask(<addr>:<port>[:<ignore>])`

Übermittelt den Request an den Policyservice auf <addr>:<port>. Entspricht die Antwort dem pcre-Pattern <ignore> geht es im Regelwerk weiter. Andernfalls wird der Wert an Postfix übermittelt.

Bsp: `id=R001; size>=100000; action=ask(127.0.0.1:10031:^dunno$)`

Aktionen (4/4)

- postfwd prüft nur die Gültigkeit von postfwd Aktionen. Es muss also sorgsam geprüft werden, ob Postfix z.B. mit einer zurückgegebenen Restriction Class etwas anfangen kann.

- Eine Übersicht der gültigen postfwd Aktionen findet sich unter http://www.postfwd.org/doc.html#actions

Statistiken und Caching (1/2)

- postfwd vermerkt in regelmäßigen Abständen Statistiken zur Nutzung im Protokoll (syslog). Das entsprechende Intervall lässt sich durch Angabe von --summary setzen (default=600s, 0 deaktiviert):

```
[STATS] postfwd2::policy 0.18RC1: 110 requests since 0 days, 0:30:09 hours
[STATS] Requests: 0.13/min last, 0.17/min overall, 0.33/min top
[STATS] Dnsstats: 0.50/min last, 0.61/min overall, 2.17/min top
[STATS] Hitrates: 75.5% ruleset, 13.6% parent, 10.9% child, 0.0% rates
[STATS] Timeouts: 0.0% (0 of 387 dns queries)
[STATS]   109 matches for id:  COUNTRY
[STATS]    76 matches for id:  T-HELO-1
[STATS]    59 matches for id:  W-HOST
[STATS]    27 matches for id:  DEBUG_MX_ADDR
[STATS]    27 matches for id:  DEBUG_NS_ADDR
 ...
```

Statistiken und Caching (2/2)

- Bei Mails mit mehreren Empfängern und Prüfung in smtpd_recipient_restrictions wird pro Empfänger ein Policy Delegation Request erzeugt. Um die Verarbeitung zu beschleunigen und das Sysem zu entlasten, verfügt postfwd über einen internen Cache für Requests und DNS Abfragen. Der Default Wert für den Request Cache liegt bei 600s. Steuerung erfolgt mittels --cache=300 (0 deaktiviert).

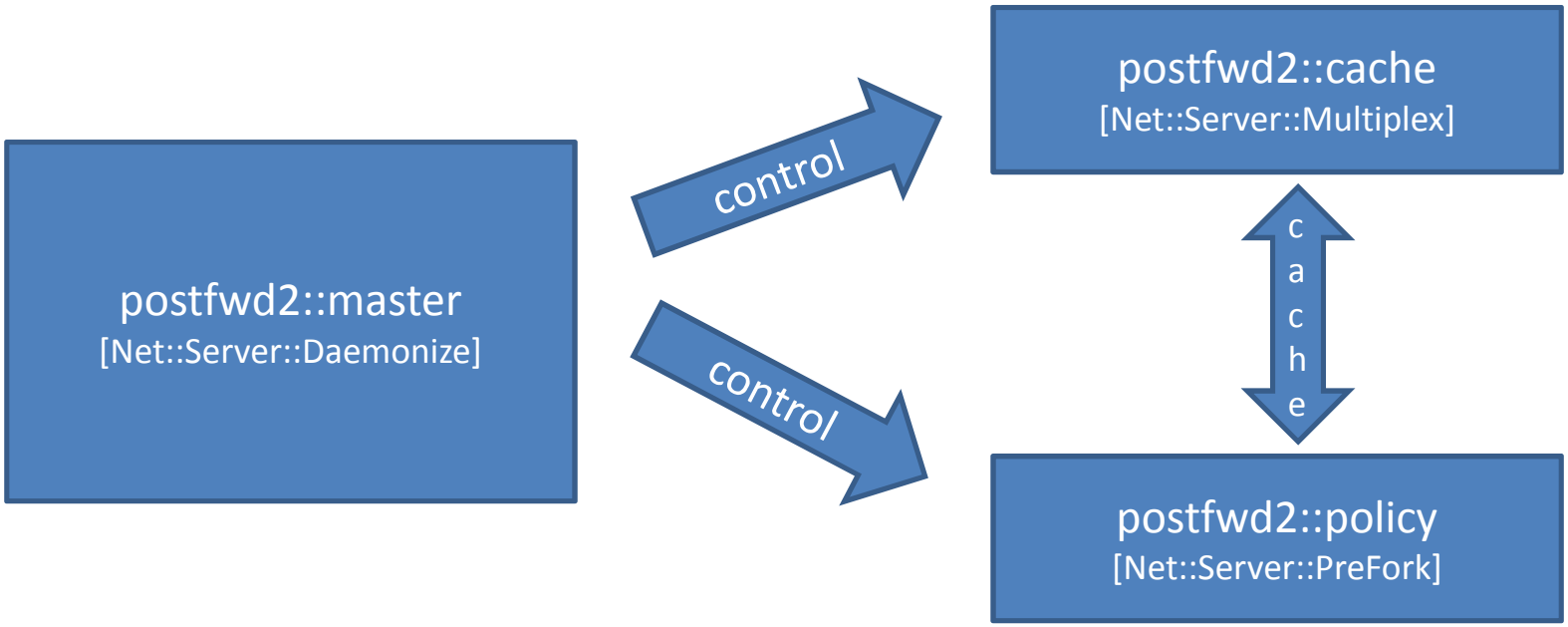
- Um einen Request im Cache vermerken zu können, wird eine Cache-ID erzeugt. Als default werden verschiedene Elemente eines Requests verwenden. Besser und vor allem performanter ist jedoch das Setzen eigener Kriterien mittels der --cacheid Option:

  ```
  # postfwd –f /etc/postfwd.cf –cacheid=client_address,recipient_domain
  ```

  führt dazu, dass nach der ersten Evaluierung des Regelwerkes, alle weiteren Requests mit gleicher IP und Empfängerdomain das gleiche Resultat auslösen – ohne erneute Evaluierung des Regelwerkes.

Ausblick: postfwd2 (1/3)

Postfwd2 folgt einer neuen Architektur aus 3 Bestandteilen.



Ausblick: postfwd2 (2/3)

- postfwd2::master stoppt und startet die anderen Bestandteile. Außerdem bezieht er von den Kindprozessen Daten zu den Statistiken. Er verfügt auch über einen Watchdog Timer, der nicht reagierende Kindprozesse ggf neu startet.

- postfwd2::cache dient als gemeinsamer Cache für Requests, DNS und Ratelimits aller postfwd2::policy Prozesse.

- postfwd2::policy bearbeitet jede Anfrage von Postfix in einem eigenen Kindprozess. Als preforking Server startet er aus Performancegründen einen Pool solcher Kindprozesse bei Programmstart und sorgt dafür, dass während der Verarbeitung immer eine gewisse Anzahl an spare children zur Verfügung stehen. Ein gravierender Fehler während der Verarbeitung kann also maximal dazu führen, dass die Bearbeitung des aktuellen Requests abbricht, nicht aber das gesamte Programm.

Ausblick: postfwd2 (3/3)

- postfwd1 und postfwd2 verwenden die gleiche Syntax zur Definition von Regeln. Bestehende Regelwerke können also ohne Anpassungen ausgetauscht werden.

- postfwd2 führt Debug-Klassen ein, mit denen sich bestimmte Vorgänge gezielter untersuchen lassen (z.B. –debug=config,cache)

- postfwd2 befindet sich noch im DEVEL Status. Ursache dafür ist v.a. die noch nicht vorhandene Implementation von Ratelimits. Die weiteren Features sind jedoch bereits verfügbar und die Version ist bereits auf zahlreichen Produktivsystemen im Einsatz (Linux, Solaris, FreeBSD)

- Die Leistung von postfwd2 skaliert mit der Anzahl der CPUs / Kerne

Weiterführende Informationen

- Postfwd Main Page
  http://www.postfwd.org

  Quickstart Guide
  http://www.postfwd.org/quick.html

  Dokumentation
  http://www.postfwd.org/doc.html

- Postfix SMTP Access Policy Delegation
  http://www.postfix.org/SMTPD_POLICY_README.html

Danke für Ihre Aufmerksamkeit!

Homepage

http://www.postfwd.org

E-Mail

info@postfwd.org