Hintergründe und Basiswissen

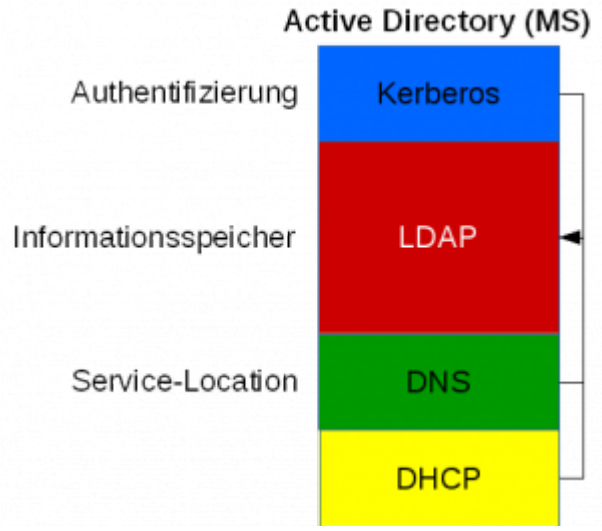
Diese Seite schließt eine Lücke im Wiki. Sie soll Hintergrund und Basiswissen vermitteln, welches dabei hilft sich an der Entwicklung des invis-Servers zu beteiligen bzw. genutzte Techniken und Konfigurationen zu verstehen.

invis-Server & Active Directory

„Active Directory“ ist das Herz des invis-Servers ab Version 10.0, das erklärt aber kaum was es genau ist.

Microsoft Actice Directory

„Active Directory“ ist eine von Microsoft entwickelte Kombination aus Kerberos, LDAP-Verzeichnis und Domain-Name-Service. Dabei dient Kerberos der Benutzer-Authentifizierung und ermöglicht sogenanntes „Single-Sign-On“, das LDAP-Verzeichnis dient als zentraler Informationsspeicher und DNS wird für eine Technik namens „Service Location“ genutzt. Eingeführt wurde „Active Directory“ mit Einführung des „Microsoft Servers 2000“.



Single-Sign-On (SSO) ermöglicht, dass nur eine Anmeldung am System auch als Autorisierung für alle im Netzwerk angebotenen Dienste gilt. D.h. es genügt beispielsweise sich am eigenen PC mit Benutzernamen und Passwort anzumelden um danach ohne weitere Angabe von Benutzernamen oder Passwort auch Zugriff auf Dienste wie beispielsweise einen Exchange-Server zu erhalten.

Kerberos seinerseits verwaltet keinerlei eigenen Datenbestand. Die Daten auf die Kerberos zurückgreift werden in einem sogenannten LDAP-Verzeichnis abgelegt. Das LDAP-Verzeichnis stellt somit den zentralen Informationsspeicher des Active Directories dar. LDAP (Lightweight Directory Access Protocol) steht dabei sowohl für das Netzwerkprotokoll über welches auf die Informationen im Verzeichnis zugegriffen wird, als auch den Informationsspeicher selbst. Der Datenspeicher arbeitet „objektorientiert“ und somit gänzlich anders als eine relationale Datenbank. Der objektorientierte Ansatz ermöglicht es Netzwerk- oder Organisationsstrukturen eins zu ein im Datenspeicher abzubilden.

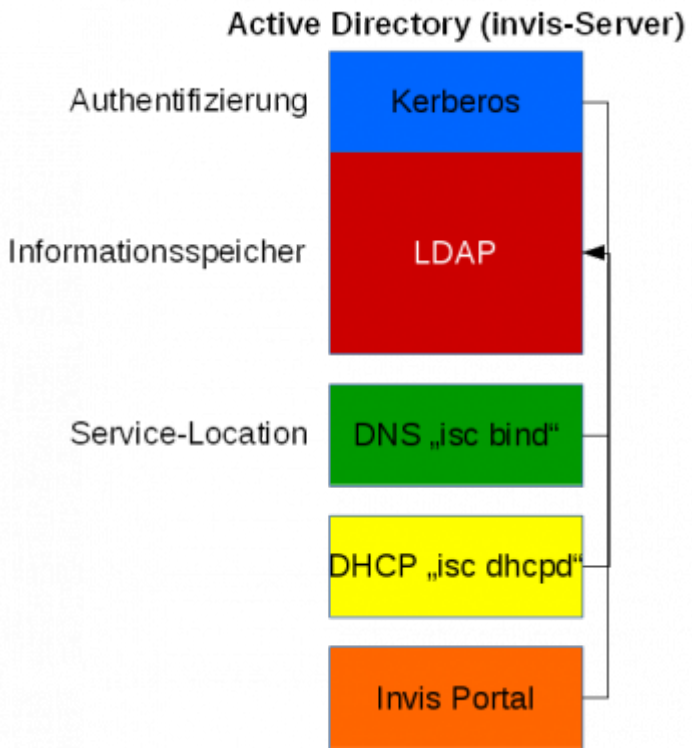
Die letzte Komponente „Domain Name Service“ (DNS) wird neben der normalen Namensauflösung für

Service Location genutzt. Service Location ist eine Technik mit der Clients im Netzwerk die Server finden können, die bestimmte Dienste anbieten. Im AD dient sie dazu den Clients mitzuteilen auf welchem Server Kerberos- und LDAP-Dienst laufen.

Zusätzlich kann auch ein DHCP-Dienst an das AD angelagert werden. Dabei nutzt der DHCP-Dienst ebenfalls das LDAP-Verzeichnis als Informationsspeicher.

Active Directory auf invis-Server

Die Verwendung eines LDAP-Verzeichnisses als zentralen Informationsspeicher für Benutzerverwaltung, DNS- & DHCP-Server sowie weitere Zwecke zu nutzen ist für invis-Server nicht neu. Mit openLDAP verfügte auch der invis Classic schon immer über ein LDAP-Verzeichnis. Was fehlte waren Kerberos und Service Location.



Mit Einführung von Samba 4.0 brachte Samba eine eigene Implementation von Microsofts Active Directory mit. Dies ermöglichte uns den Umstieg von openLDAP auf Active Directory. Um alle Informationen die der invis Classic im LDAP speicherte auch im AD speichern zu können mussten wir eigene Schemaerweiterungen einspielen und darüber hinaus auch den von uns verwendeten DHCP-Server des ISC (Internet Systems Consortium) patchen.

Als DNS-Server verwenden wir nicht den „Samba Internal DNS“, sondern weiterhin „bind“ (ebenfalls vom ISC). Dies wird allerdings von Samba direkt unterstützt. Die Anbindung erfolgt hier über einen DLZ-Treiber (Dynamic Loadable Zone) den Samba selbst mitbringt. Nebenbei bemerkt ist es auch in Microsoft AD-Umgebungen möglich den DNS-Dienst auszulagern, ich habe das mit „ISC bind“ bereits erfolgreich getestet.

Für die Benutzerverwaltung haben sich die Vorzeichen umgekehrt. Beim invis Classic wurden im LDAP prinzipiell POSIX-kompatible Benutzerkonten angelegt, die um Windows- respektive Samba-Attribute ergänzt wurden. Mit Active Directory bilden jetzt Windows-Benutzerkonten die Basis, die um POSIX-Attribute erweitert werden. Da Microsoft dies selbst unter Verwendung der eigenen Erweiterung „Services for UNIX“ (SFU) anbietet, bricht es nicht die Kompatibilität zur MS-Welt. Unter Samba wird diese Ergänzung „RFC 2307 Erweiterung“ genannt.

Die Anbindung von Linux-Clients wird über eine Kombination aus Samba und SSSD (System Security Services Daemon) realisiert. Dabei wird Samba genutzt um (genau wie Windows Clients) der Domäne

beizutreten und SSSD ist für die Abbildung der Benutzerkonten via PAM auf Linux-Systemen sowie die Benutzeranmeldung selbst zuständig. Selbstverständlich wird auch hier Kerberos verwendet.

Genau wie es für Windows-Domänenmitglieder möglich ist, ermöglicht der SSSD auch für Linux-Clients Offline-Logins an der Domäne.

Grundsätzlich beherrschen invis-Server damit auch Single Sign On, leider haben wir dass noch nicht für die Dienste des invis-Servers umgesetzt.

Active Directory und Firewall

Die vielen an AD beteiligten Netzwerkprotokolle erfordern ein besonderes Augenmerk auf eine zwischen geschaltete Firewall. invis-Server nutzen beispielsweise immer die „SuSEfirewall2“ auch zwischen dem Server selbst und dem lokalen Netzwerk (interne Zone).

Die folgende Tabelle gibt einen Überblick über die beteiligten Protokolle und zugehörigen Ports:

Protokoll	Transportprotokoll	Port	Bemerkung
Kerberos	TCP & UDP	88	Benutzerauthentifizierung
LDAP	TCP & UDP	389	Daten-Backend, Benutzerveraltung, DNS, DHCP und invis-Portal
LDAPs	TCP	636	s.o.
DNS	TCP & UDP	53	Namensauflösung und Service-Location
	TCP	1024	Zugriff auf DNS-Daten mittels RSAT
GC	TCP	3268	Global Catalog
GC	TCP	3269	Global Catalog SSL
SMB/CIFS	TCP & UDP	445	Zugriff auf sysvol Freigabe
RPC	TCP	135	Früheres Microsoft Messaging Protokoll, wird genutzt für die Replikation der sysvol Freigabe. Dies wird von Samba noch nicht unterstützt.
	TCP & UDP	464	Replication, User and Computer Authentication, Trusts

ISC DHCP mit ActiveDirectory

Um den DHCP-Server des ISC mit einem Active Directory als Backend zu verwenden, muss dieser so gepatcht werden, dass er andere LDAP-Attribut- und Objektklassen-Namen akzeptiert. Dies ist notwendig, da Microsoft für seinen DHCP-Server teils identische Namen für Attribute und Objektklassen verwendet. Ohne das Patchen führt der Versuch einen ISC DHCP-Server Datenbestand aus einem OpenLDAP Verzeichnis in ein Active Directory zu migrieren zu „Objectclass Violations“ die nicht zu beheben sind.

LDAP Patch erzeugen

Den Patch zu erstellen ist etwas umständlich, da im openSUSE-Paket bereits Patches auf die von uns zu patchenden Dateien angewendet werden. D.h. um unseren Patch zu erzeugen müssen zunächst die anderen Patches auf den Quellcode angewendet werden.

Zu patchen sind folgende Dateien:

- server/ldap.c
- contrib/ldap/README.ldap
- contrib/ldap/dhcpd-conf-to-ldap
- contrib/ldap/dhcp.shema

Im ersten Schritt müssen aus dem Repository „network:dhcp“ (https://build.opensuse.org/package/show/network:dhcp/dhcp) des openSUSE Buildservice das Quellcode-Archiv (dhcp-4.x.y.tar.gz), sowie die beiden Patches „0007-dhcp-4.2.6-ldap-mt01.patch“ und „0022-dhcp-4.2.x-contrib-conf-to-ldap-reorder.886094.patch“ heruntergeladen werden.

Das Quellcode-Archiv ist in ein Arbeitsverzeichnis zu extrahieren. Danach sind die beiden heruntergeladenen Patches darauf anzuwenden:

```
heinzb@knurps:~/baustelle/dhcp-4.2.6> patch -p1 server/ldap.c 0007-
dhcp-4.2.6-ldap-mt01.patch
heinzb@knurps:~/baustelle/dhcp-4.2.6> patch -p1 contrib/ldap/dhcpd-conf-to-
ldap 0022-dhcp-4.2.x-contrib-conf-to-ldap-reorder.886094.patch
```

Treten dabei Fehler auf, sind diese in einer auf **.rej** endenden Datei im Verzeichnis der zu patchenden Datei zu finden. Sind die Fehler überschaubar, lassen sich die Änderungen auch manuell durchführen.

Jetz sind per suchen und ersetzen der Attributnamen und Objektklassen mit ***sed*** neue Dateien mit den gewünschten Änderungen zu erzeugen:

```
heinzb@knurps:~/baustelle/dhcp-4.2.6> cat server/ldap.c | sed /dhcp[A-
Z]/s/dhcp/iscDhcp/g > server/ad.ldap.c
heinzb@knurps:~/baustelle/dhcp-4.2.6> cat contrib/ldap/README.ldap | sed
/dhcp[A-Z]/s/dhcp/iscDhcp/g > contrib/ldap/ad.README.ldap
heinzb@knurps:~/baustelle/dhcp-4.2.6> cat contrib/ldap/dhcpd-conf-to-ldap |
sed /dhcp[A-Z]/s/dhcp/iscDhcp/g > contrib/ldap/ad.dhcpd-conf-to-ldap
heinzb@knurps:~/baustelle/dhcp-4.2.6> cat contrib/ldap/dhcp.schema | sed
/dhcp[A-Z]/s/dhcp/iscDhcp/g > contrib/ldap/ad.dhcp.schema
```

Jetzt werden mit ***diff*** einzelne Patches für jede der Dateien erzeugt:

```
heinzb@knurps:~/baustelle/dhcp-4.2.6> diff -u server/ldap.c server/ad.ldap.c
> ../ldap.c.patch
heinzb@knurps:~/baustelle/dhcp-4.2.6> diff -u contrib/ldap/README.ldap
contrib/ldap/ad.README.ldap > ../README.ldap.patch
heinzb@knurps:~/baustelle/dhcp-4.2.6> diff -u contrib/ldap/dhcpd-conf-to-
ldap contrib/ldap/ad.dhcpd-conf-to-ldap > ../dhcpd-conf-to-ldap.patch
heinzb@knurps:~/baustelle/dhcp-4.2.6> diff -u contrib/ldap/dhcp.schema
contrib/ldap/ad.dhcp.schema > ../dhcp.schema.patch
```

Jetzt müssen noch die Kopfzeilen der Einzelpatches angepasst werden. Aus:

```
--- server/ldap.c       2015-03-26 14:24:44.905456906 +0100
+++ server/ad.ldap.c    2015-03-26 17:12:52.783318249 +0100
```

wird:

```
--- a/server/ldap.c       2015-03-26 14:24:44.905456906 +0100
+++ b/server/ldap.c    2015-03-26 17:12:52.783318249 +0100
```

Danach werden alle Einzelpatches zum Gesamtpatch zusammengesetzt:

```
heinzb@knurps:~/baustelle> cat ldap.c.patch > 0099-AD-LDAP.patch
heinzb@knurps:~/baustelle> cat README.ldap.patch >> 0099-AD-LDAP.patch
heinzb@knurps:~/baustelle> cat dhcpd-conf-to-ldap.patch >> 0099-AD-
LDAP.patch
heinzb@knurps:~/baustelle> cat dhcp.schema.patch >> 0099-AD-LDAP.patch
```

Danach den Patch in unser Testing-Repositorie hochladen und die Spec-Datei des dhcp-Paketes anpassen:

Ab Zeile 138 (ca.)

```
# invis-server.org Stefan Schaefer
Patch99:        0099-AD-LDAP.patch
```

Ab Zeile: 282 (ca.)

```
%if %{with_ldap}
# Stefan Schaefer - invis-server.org
%patch99 -p1
%endif
%endif
## clean up after patching
find . -type f -name \*.orig\* -exec rm -f {} \;
find . -type f -name \*.rej\* -exec rm -f {} \;
```

Die Zeilenangaben sind natürlich nicht genau, das sich das Spec-File mit jeder neuen Version ändern kann. Wichtig ist nur, dass unser Patch als letzter angewendete wird.

Paketbenennung und Konfliktverhalten

Um nicht mit den regulären DHCP-Paketen in den SUSE Repositories in Konkurrenz zu geraten, haben wir unsere Pakete umbenannt. Sie tragen jetzt die Vorsilbe „invis“:

- invisdhcp
- invisdhcp-server
- invisdhcp-client
- invisdhcp-relay
- invisdhcp-devel
- invisdhcp-doc

Die Umbenennung erfolgt im Specfile. Weiterhin wurde in den Paketen die Direktive „Conflict“ im Specfile verwendet. Dadurch können nur entweder die ungepatchten oder die gepatchten Pakete auf einem System installiert werden. Auch dies wird im Specfile eingetragen:

Namensdefinition ab Zeile 35:

```
Name:           invisdhcp
%define originname      dhcp
....
Source0:        dhcp-%{isc_version}.tar.gz
Source1:        dhcp-%{isc_version}.tar.gz.asc
Source2:        %{originname}.keyring
```

Die Variable %originname musste hinzugefügt werden, da der Originalname weiterhin beim Paketbau benötigt wird.

Conflict & Requires Direktiven ab Zeile 146

```
Conflicts: dhcp

%package server
Summary:        ISC DHCP Server
Group:          Productivity/Networking/Boot/Servers
Requires:       invisdhcp = %{version}
Requires:       net-tools
Conflicts:  dhcp-server
PreReq:         %insserv_prereq %fillup_prereq /bin/cat /bin/mkdir /bin/cp
/usr/sbin/useradd

%package client
Summary:        ISC DHCP Client
Group:          Productivity/Networking/Boot/Clients
Requires:       /sbin/arping
Requires:       /usr/bin/host
Requires:       invisdhcp = %{version}
Requires:       iproute2
Requires:       net-tools
Conflicts:  dhcp-client
PreReq:         %insserv_prereq %fillup_prereq /bin/cat /bin/mkdir /bin/cp
/bin/grep

%package relay
Summary:        ISC DHCP Relay Agent
Group:          Productivity/Networking/Boot/Servers
Requires:       invisdhcp = %{version}
Requires:       net-tools
Conflicts:  dhcp-relay
PreReq:         %insserv_prereq %fillup_prereq /bin/cat /bin/mkdir /bin/cp

%package devel
Summary:        Header Files and Libraries for dhcpctl API
Group:          Development/Libraries/C and C++
Requires:       invisdhcp = %{version}
Conflicts:  dhcp-devel
Conflicts:      bind-devel
```

```
%if %{with_doc_package}

%package doc
Summary:        Documentation
Conflicts:  dhcp-doc
Group:          Productivity/Networking/Boot/Servers
%endif
```

Wie gezeigt sind Anpassungen in allen Einzelpaketen erforderlich. Mittels Conflicts und Requires werden die erforderlichen Paketabhängigkeiten gelöst.

DNS-Server bind und LDAP (invis Classic)

Schon von Anfang an setzen wir ein LDAP-Verzeichnis als Daten-Backend für den DNS-Server *bind* ein. Der zugrunde liegende Patch, welchen wir schon seit einiger Zeit selbst in die Software-Pakete einbauen müssen, wird inzwischen kaum noch weiter entwickelt. Statt dessen gehört der sogenannte DLZ (dynamic loadable zone) Treiber seit etwa openSUSE 11.2 (oder so) fest zum Funktionsumfang der openSUSE-Pakete. Die Nutzung dieses Treibers würde uns eine Menge Arbeit ersparen, da wir keine eigenen *bind* Pakete mehr im openSUSE-Buildservice pflegen müssten.

Schema-Definitionen

Jede Form der Datenspeicherung in einem LDAP-Verzeichnis bedarf der Definition von Attributen und Objektklassen (Tabellen- und Felddefinitionen einer SQL-Datenbank nicht ganz unähnlich). Diese Definitionen werden in Form sogenannter Schema-Dateien getroffen und haben Server-weit Gültigkeit. Zu finden sind Sie im Verzeichnis

```
/etc/openldap/schema
```

Der Aufbau der LDAP-Schemas (kein Witz, es schreibt sich so) „dlz.schema“ für den DLZ-Treiber und „dnszone.schema“ für den alten SDB-LDAP-Patch unterscheiden sich grundsätzlich voneinander.

Einige Unterschiede im Überblick:

- Allen Attributen und Objektklassen des DLZ-Schemas ist zur Vermeidung von Namensdopplungen die Silbe „dlz“ vorangestellt.
- Während das *DNSZone-Schema* lediglich die Objektklasse „dNSZone“ kennt über die alle DNS-Objekte dargestellt werden, verfügt das *DLZ-Schema* über Objektklassen für die verschiedenen DNS-Record-Typen. Dies dürfte etwas verwirrend wirken, da es für die verschiedenen DNS-Records auch entsprechende Attributdefinitionen gibt.
- Nach dem alten Schema beginnt jedes DNS-Objekt im LDAP mit dem Attribut „relativeDomainName“, welches als Schlüssel-Attribut angesehen werden kann. Im DLZ-Schema nimmt das Attribut „dlzRecordID“ diese Rolle ein. Eine Kennzeichnung des Objekts wird dabei erst über das Folge-Attribut erreicht: „dlzRecordID=05,dlzHostName=pc-buchhaltung,…“
- Während alle kennzeichnenden Informationen eines SOA-Records beim alten Schema im Attribut „sOARecord“ zusammengefasst wurden, kennt das DLZ-Schema für jeden Wert ein eigenes Attribut: z.B. „dlzSerial“, „dlzRefresh“, usw.
- Die Daten der MX-Records sind beim DLZ-Schema nicht Teil des SOA-Eintrages, sondern werden

als eigene LDAP-Objekte angelegt.
- Das DLZ-Schema kennt weder Attribute noch Objektklassen für SRV-, mINFO, hINFO, SIG-, KEY-, sowie einige weitere DNS-Record-Typen. Diese können entweder über die Objektklasse „dlzGenericRecord“ oder eigene Erweiterungen des Schemas realisiert werden. Für Letzteres ist mit „1.3.6.1.4.1.18420.1.3.X“ im Schema ein eigener Object-Identifier (OID) vorgesehen.
- Die meisten Attribute des DLZ-Schemas sind als Single-Values gekennzeichnet. D.H. gehören etwa zu einem Pointer-Record mehrere Hostnamen, können diese nicht in einem Objekt zusammengefasst werden, sondern es muss für jeden Hostnamen ein eigenes PTR-Objekt erzeugt werden.

Aufbau der LDAP-Objekte

Um die Unterschiede im Aufbau der jeweiligen LDAP-Struktur zu verdeutlichen werden im folgenden einige Standard-Einträge gegenübergestellt.

SOA-Record

- dNSZone-Schema

```
dn: relativeDomainName=@,ou=invis-net.loc,ou=forward,ou=zone.master,ou=DNS-
Server,dc=invis-net,dc=loc
dNSClass: IN
dNSTTL: 3600
mXRecord: 10 mail.invis-net.loc.
nSRecord: ns.invis-net.loc.
objectClass: dNSZone
objectClass: top
relativeDomainName: @
sOARecord: ns.invis-net.loc. root.invis-net.loc. 2006260342 3600 1800 604800
86400
zoneName: invis-net.loc
```

- DLZ-Schema

```
dn: dlzRecordID=11,dlzHostName=@,dlzZoneName=invis-
net.loc,ou=zone.master,ou=DNS-Server,dc=invis-net,dc=loc
objectclass: dlzSOARecord
dlzRecordID: 11
dlzHostName: @
dlzType: soa
dlzSerial: 2006260342
dlzRefresh: 3600
dlzRetry: 1800
dlzExpire: 604800
dlzMinimum: 86400
dlzAdminEmail: root.invis-net.loc.
dlzPrimaryns: ns.invis-net.loc.
dlzTTL: 10
```

```
dn: dlzRecordID=5,dlzHostName=@,dlzZoneName=invis-
server.loc,ou=zone.master,ou=DNS-Server,dc=invis-server,dc=loc
```

```
objectclass: dlzMXRecord
dlzRecordID: 5
dlzHostName: @
dlzType: mx
dlzData: mail
dlzPreference: 10
dlzTTL: 10
```

A-Record

- dNSZone-Schema

```
# A-Record des Servers
dn: relativeDomainName=invis5,ou=invis-
net.loc,ou=forward,ou=zone.master,ou=DNS-Server,dc=invis-net,dc=loc
aRecord: 192.168.220.10
dNSClass: IN
dNSTTL: 86400
objectClass: dNSZone
objectClass: top
relativeDomainName: invis5
zoneName: invis-net.loc
```

- DLZ-Schema

```
dn: dlzRecordID=14,dlzHostName=invis5,dlzZoneName=invis-
net.loc,ou=zone.master,ou=DNS-Server,dc=invis-net,dc=loc
objectclass: dlzARecord
dlzRecordID: 14
dlzHostName: invis5
dlzType: a
dlzIPAddr: 192.168.220.10
dlzTTL: 86400
```

PTR-Record

- dNSZone-Schema

```
dn: relativeDomainName=10,ou=220.168.192.in-
addr.arpa,ou=reverse,ou=zone.master,ou=DNS-Server,dc=invis-net,dc=loc
pTRRecord: invis5.invis-net.loc.
pTRRecord: ns.invis-net.loc.
pTRRecord: mail.invis-net.loc.
dNSClass: IN
dNSTTL: 86400
objectClass: dNSZone
objectClass: top
# Test ohne Leerzeichen
relativeDomainName:10
zoneName: 220.168.192.in-addr.arpa
```

- DLZ-Schema

```
dn: dlzHostName=10,dlzZoneName=220.168.192.in-
addr.arpa,ou=zone.master,ou=DNS-Server,dc=invis-net,dc=loc
objectclass: dlzHost
dlzHostName: 10
```

Die Begründung für diesen Eintrag finden Sie im nächsten Abschnitt.

```
dn: dlzRecordID=15,dlzHostName=10,dlzZoneName=220.168.192.in-
addr.arpa,ou=zone.master,ou=DNS-Server,dc=invis-net,dc=loc
objectclass: dlzPTRRecord
dlzRecordID: 15
dlzHostName: 10
dlzType: ptr
dlzData: invis5.invis-net.loc.
dlzTTL: 86400
```

```
dn: dlzRecordID=16,dlzHostName=10,dlzZoneName=220.168.192.in-
addr.arpa,ou=zone.master,ou=DNS-Server,dc=invis-net,dc=loc
objectclass: dlzPTRRecord
dlzRecordID: 16
dlzHostName: 10
dlzType: ptr
dlzData: mail.invis-net.loc.
dlzTTL: 86400
```

```
dn: dlzRecordID=17,dlzHostName=10,dlzZoneName=220.168.192.in-
addr.arpa,ou=zone.master,ou=DNS-Server,dc=invis-net,dc=loc
objectclass: dlzPTRRecord
dlzRecordID: 15
dlzHostName: 10
dlzType: ptr
dlzData: ns.invis-net.loc.
dlzTTL: 86400
```

LDAP-Struktur

Auch die Struktur des LDAP-Zweiges für den DNS-Server sollte sich mit Umstieg auf das DLZ-Schema ändern.

Bisher wurden beim invis-Server die DNS-Einträge nach Zone, Zonen-Typ und „Richtung“ (Vorwärts/Rückwärts) sortiert. Dies war in sich eher etwas unglücklich. Mit einem Umstieg werden wir uns an die Empfehlung des „Bind DLZ“ Projekts halten und lediglich nach Zone und Host sortieren. Eine Unterscheidung nach Vorwärts- oder Rückwärtsauflösung ist nicht generell nicht notwendig, da dies bereits über die Zone selbst klar wird.

Der Basisknoten bleibt hingegen unverändert:

```
ou=DNS-Server,ou=domain,ou=tld
```

Dem wurde beim alten Schema noch der Zonentyp (Master oder Slave), die Richtung der Auflösung sowie der Zonenname vorangestellt.

```
ou=domain.tld,ou=forward,ou=zone.master,...
```

Zukünftig wird sich dies Ändern. Zunächst ist der Zonenname hier mit einem eigenen Attribut gekennzeichnet, dafür eine eigene „Organizational Unit“ (ou) in den DN (Distinguished Name) eines DNS-Objektes einzufügen ist also obsolet. Weiterhin entfällt wie bereits erwähnt die Unterscheidung zwischen Vorwärts- und Rückwärtsauflösung, etwas worauf wir auch schon früher hätten verzichten können. Zu guter Letzt fügen wir der DLZ-Empfehlung folgend allerdings bei Bedarf den Hostnamen als zusätzlichen Knoten ein. Dies ist dann sinvoll, wenn etwa zu einem Host mehrere Einträge, seien es A- oder PTR-Records, folgen:

Also, entweder:

```
dlzZoneName=domain.tld,ou=zone.master,....
```

oder:

```
dlzHostname=host,dlzZoneName=domain.tld,ou=zone.master,....
```

Letzteres eben im Falle mehrerer Records zu einem Host.

Nameserver Konfiguration

Grundsätzlich hat sich auch die Konfiguration des Nameservers bind in Sachen Zugriff auf das LDAP-Backend geändert. Statt getrennte Zonenkonfigurationen für Forward und Reverse Zonen gibt es nur noch einen Eintrag in der Datei named.conf.

```
## LDAP Backend mit DLZ-Schema
dlz "ldap zone" {
        database "ldap 2
           v3 simple {uid=Admin,ou=DNS-Server,dc=invis-server,dc=loc}
{passwort} 127.0.0.1
           ldap:///dlzZoneName=$zone$,ou=zone.master,ou=DNS-Server,dc=invis-
server,dc=loc???objectclass=dlzZone
ldap:///dlzHostName=$record$,dlzZoneName=$zone$,ou=zone.master,ou=DNS-
Server,dc=invis-
server,dc=loc?dlzTTL,dlzType,dlzPreference,dlzData,dlzIPAddr,dlzPrimaryNS,dl
zAdminEmail,dlzSerial,dlzRefresh,dlzRetry,dlzExpire,dlzMinimum?sub?objectcla
ss=dlzAbstractRecord";
};
```

Wichtig bei der Bearbeitung der Einträge ist, dass die einzelnen LDAP-URLs nicht umbrochen werden und keine Leerzeichnen enthalten dürfen. Die notwendigen Anpassungen dieses Eintrags auf die jeweilige Umgebung beschränken sich auf den Bind-DN, mit dem sich der Dienst am LDAP-Verzeichnis anmeldet und den DN, der Zonen in der ersten LDAP-URL.

Daneben ist ggf. noch die Möglichkeit von Interesse bind mit mehreren Threads gleichzeitig auf das

LDAP-Verzeichnis zugreifen zu lassen. Dies ist vor allem bei hochfrequentierten Nameservern ein gutes Mittel die Performance des Nameservers zu steigern. In kleineren Umgebungen spielt die keine große Rolle.

Im Beispiel wurden zwei gleichzeitige Threads ermöglicht. Hierfür steht die Ziffer (2) am Beginn der Definition des Daten-Backends. Vorgegeben ist der Wert 1, da die Erhöhung nur dan funktioniert, wenn das zugrunde liegende System Multithreading-fähig ist. Bei modernen Linux-Systemen ist dies in der Regel der Fall.

DDOS / Amplifier Attacke auf bind

Noch ein schönes Thema für die Knowledgebase, diesmal im Hinblick auf das Thema Rootserver.

Die Attacke

Beginnend etwa am 25. Oktober 2012 geriet einer meiner produktiv genutzten Rootserver unter Beschuss. Aufgefallen ist mir das Ganze nur, weil invis-Server meiner Kunden immer schlechter via Internet erreichbar waren. Zunächst ließ mich mein Monitoring-System vermuten, dass es Probleme mit den DSL-Anbindungen bei den Kunden gibt. Es stellte sich aber heraus, dass der DNS-Server der für meine Kunden dynamisches DNS (DDNS – ist im Root-Server Buch beschrieben) betreibt, attackiert wurde und dadurch seinen Dienst nicht mehr zuverlässig erledigen konnte.

Im Logfile tauchten massiv immer wieder die gleichen Anfragen auf:

```
Nov 15 15:11:02 rs2 named[2637]: client 77.xxx.xxx.xxx#64253 (.): query: .
IN ANY +E (178.xx.xx.xx)
```

Derartige Anfragen sind unschön. Sie nutzen aus, dass der DNS Server, so er ein offener Resolver ist, auch Anfragen beantwortet, die außerhalb seines eigenen Datenbestandes liegen. Im Beispiel werden schlicht alle (ANY) Informationnen zur Rootzone (**.**) erfragt.

Eine solche Anfrage erzeugt Antworten die weit größer (bis zum Faktor 60, laut einigen Dokumentationen) sind als die früher üblichen 512 Byte. In den Traffic-Logs des Servers spiegelte sich das deutlich nieder. Während in Zeiten vor dem Angriff auf dem Server eingehend ca. 1 bis 2 GB und ausgehend 2 bis 3 GB pro Tag anfielen, stieg der ausgehende Traffic im Verlauf der Attacke bis auf ca. 25 GB pro Tag an. Der Nameserver (im Einsatz ist bind) stieß an die Grenzen seiner Möglichkeiten und beantwortete reguläre Anfragen nicht mehr zuverlässig.

Die Diskrepanz zwischen eingehender und bis zum Faktor 60 höherer ausgehender Datenmenge gibt dieser Form des Angriffs ihren Namen DDOS/Amplifier (Verstärker).

Da eine Anfrage im Rechenzentrum ergeben hat, dass von weiteren Attacken auf andere Server nichts bekannt ist und die Attacke auf meinen Server von immer anderen Hosts ausging, liegt der Schluss nahe, dass es sich um einen gesteuerte, zielgerichteten Angriff handelt (Distributed Denial Of Service Attack / DDOS).

Leider sind weder die Gründe dafür noch die Angreifer ohne Weiteres ermittelbar.

Gegenmaßnahmen

Für die Abwehr eines solchen Angriffes ergeben sich genau zwei Ansatzpunkte:

- **Der Dienst selbst**
- **Die Firewall**

Über die Firewall lassen sich Limits für Anfragen pro Zeit auf bestimmte Ports einrichten. Eine solche Maßnahme ist zwar wirksam um die Gesamtbelastung des Servers zu senken, sie bestraft allerdings auch diejenigen, die einen Dienst regulär nutzen.

Reduziert man also die Anfragen auf Port 53/UDP beispielsweise auf 200 pro Sekunde, lässt die Firewall eben nur noch eben diese 200 Anfragen auf den DNS-Server zu, ungeachtet dessen, ob eine Anfrage gut oder böse ist. Schöpft der Angreifer das Limit bereits aus, sind Störungen des regulären Betriebs die Folge. Prinzipiell, hat der Angreifer auch auf diese Weise sein Ziel schon erreicht. Der einzig messbare Erfolg dieser Strategie ist die Reduktion des Traffics.

Hier noch beispielhaft eine entsprechende Firewall-Regel:

```
linux:~ # iptables -A INPUT -p UDP -i eth0 --sport 1024: --dport 53 -m limit
--limit 200/s -j ACCEPT
```

Besser ist also die Abwehr am Dienst selbst oder eine Kombination mehrerer Maßnahmen.

Vielfach ist im Internet zu lesen, dass derartige Angriffe vor allem durch sogenannte „open resolver“ begünstigt wird. Ein „open resolver“ ist ein Nameserver, der für alle DNS Anfragen generell zur Verfügung steht. Einen DNS-Server so zu konfigurieren, dass er nur noch Fragen zu seinem eigenen Datenbestand, also seinen Zonen beantwortet führt zwar direkt zum Ziel, ist aber dann nicht möglich wenn es beispielsweise darum geht Forward-Nameserver für die eigenen Kunden zu sein.

Ist es nicht erforderlich allgemeine DNS-Anfragen zu beantworten lässt sich der oben beschriebene Angriff leicht abwehren. In der Datei „/etc/named.conf“ muss in der Options-Sektion lediglich folgender Eintrag eingefügt werden:

```
options {
...
        # Rekursion verbieten
        recursion no;
...
};
```

Soll der Nameserver allerdings öffentlich für alle Anfragen zur Verfügung stehen, kommt diese Abwehrstrategie nicht in Frage.

Es bietet sich an Clients deren Anfragen dem Muster des Angriffs entsprechen auf eine Blacklist zu setzen. Da die Angriffe allerdings von ständig neuen Hosts (vermutlich aus einem Bot-Netz) kommen, darf eine solche Blacklist aber nicht statisch sein.

Generell kann ***bind*** über die Direktive „*allow-query*“ Anfragen mit einem „Refused“ beantworten oder mit der Direktive „*blackhole*“ gänzlich ins Leere laufen lassen. Da es ja um die Vermeidung von Traffic geht bietet sich Letzteres an. Nützlich ist natürlich auch, dass ***bind*** mit Access-Control-Lists umgehen

kann.

Zur Abwehr habe ich eine externe Konfigurationsdatei erzeugt, die lediglich eine ACL namens „blackhole“ enthält. Die darin aufgeführte Liste wird dynamisch durch ein per Cron-Job regelmäßig ausgeführtes Shellscript gepflegt. Das Shellscript durchsucht beim Aufspüren der Angreifer wiederum die Datei „/var/log/messages“ nach entsprechenden Einträgen. Voraussetzung ist ein aktiviertes Query-Logging.

Das Shellscript:

```
#!/bin/bash
# Dynamisch Blacklist fuer bind erzeugen
cat /var/log/messages |grep "IN ANY +E" |cut -d " " -f7|cut -d "#" -f1| sort
-u >> /var/tmp/badhosts.tmp
cat /var/tmp/badhosts.tmp | sort -u > /var/tmp/badhosts

# Attacking Hosts acl erstellen
echo "acl blackhole {" > /etc/named.d/named.blackhole

for badhost in `cat /var/tmp/badhosts`;do
    echo -e "\t$badhost;" >> /etc/named.d/named.blackhole
done

echo -e "};\n" >> /etc/named.d/named.blackhole

# bind reload
/etc/init.d/named reload
```

Das geht sicherlich auch noch eleganter, aber auf die Schnelle erfüllte es seinen Zweck. Ausgeführt wird das Script einmal pro Minute. Häufig, aber erforderlich.

In der Datei „named.conf“ wird zunächst vor der Options-Sektion die vom Shellscript erzeugte Konfigurationsergänzung „named.blackhole“ eingebunden:

```
...
include /etc/named.d/named.blackhole
...
```

Innerhalb der Options-Sektion wird ***bind*** dann lediglich mitgeteilt, was er mit der generierten ACL machen soll:

```
options {
...
        blackhole { blackhole; 2.0.0.0/8; 224.0.0.0/3; };
        allow-query { any; };
...
};
```

Nach 4 Tagen war die Liste der Angreifer bereits über 2000 Einträge lang.

Die Kombination aus oben gezeigter Firewall-Regel und dem hier vorgestellten Blackhole-

Mechanismus lies die ausgehende tägliche Datenmenge des betroffenen Servers wieder auf Normalmaß schrumpfen. Trotzdem war dies nur ein Teilerfolg, da die Firewall-Regel auch reguläre Anfragen blockiert hat.

Die wirklich funktionierende Lösung des Problems brachte ein Patch für ***bind*** von Vernon Shryver und Paul Vixie, der ein per Client Rate-Limiting über die Konfiguration des Nameservers selbst erlaubt. Genau beschrieben wird die Funktionsweise des Patches auf der zugehörigen Internet-Seite: http://www.redbarn.org/dns/ratelimits

Im Gegensatz zum Firewall-Ansatz, wird hierbei jeder anfragende Client individuell betrachtet. D.h. Bleibt ein Client mit seinen Anfragen unter einer einstellbaren Anzahl pro Sekunde werden alle Anfragen beantwortet, überschreitet er sein Limit, antwortet ***bind*** einfach nicht mehr.

Entsprechend gepatchte ***bind*** Versionen stellen wir über unsere invis-Repositories im OpenBuildService zur Verfügung, auch noch für ältere openSUSE-Versionen (11.1, 11.2 und 11.3).

Die Konfiguration ist denkbar einfach. In der Options-Sektion der Datei „/etc/named.conf“ genügt folgender Eintrag:

```
options {
...
     rate-limit {
          responses-per-second 25;
          window 5;
     };
...
};
```

Näheres zu den Parametern der „rate-limit“ Direktive ist auf oben genannter Internet-Seite nachzulesen.

Die Kombination aus Blacklist und Rate-Limit ohne Firewall-Regel funktioniert hervorragend.

Postfix, Amavis & Dovecot

Da wir uns über kurz oder lang möglichst wieder von Cyrus-IMAP verabschieden wollen kümmern wir uns derzeit intensiv um das reibungslose Zusammenspiel von Postfix, Amavis und Dovecot. Ziel des Setups ist es:

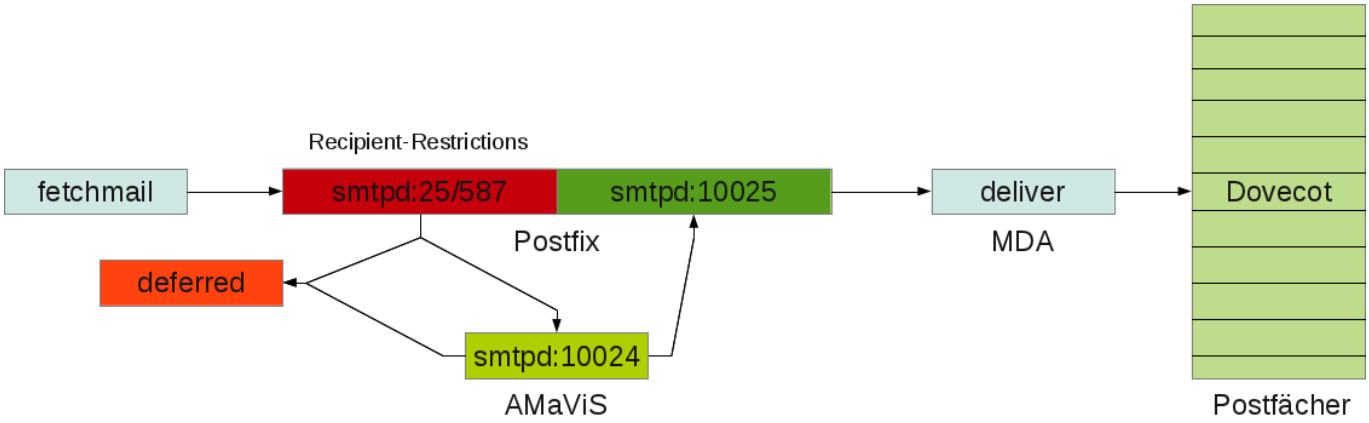
- Dovecot als SASL-Authentifizierungsbackend für Postfix zu nutzen.
- Dovecots LMTP-Dienst zur Mail-Einlieferung zu verwenden
- IMAP-ACLs / Shared Folders zu ermöglichen.
- Sieve-Filter zu ermöglichen
- Amavis als content_filter zu nutzen.

Das Ganze gilt natürlich nur, wenn kein Zarafa eingesetzt wird. Zarafa integriert den Email-Speicher und Funktionen wie Zugriffsrechte und Filterregeln, kommt also ohne externen IMAP-Server aus.

Die genannten Anforderungen ergeben ein recht komplexes Setup, daher widme ich mich hier den Hintergründen. Zunächst allerdings nur dem Teil der Einlieferung neuer Mails via ***fetchmail***.

Größte Schwierigkeit hierbei ist ein sauberes Mailrouting durch alle Komponenten. Dies beginnt bereits beim Fetchmail-Daemon. Zunächst ein schematischer Überblick über die Wege einer eingehenden Email:

Hinweis: *Die Grafik ist nicht aktuell, der Dovecot Delivery Agent fällt jetzt weg, die Einlieferung erfolgt direkt via LMTP.*



fetchmail

In der Grundkonfiguration hat der fetchmail-Daemon auf openSUSE Systemen die unangenehme Eigenschaft eingehende Emails an Adressen des Schemas „user@***localhost***.domain.tld“, statt an „user@domain.tld“ weiterzuleiten. Dies kann das gesamte Verhalten der nachfolgenden Komponenten beeinflussen. Leider nicht zum Guten.

Dies kann auf zweierlei Wegen behoben werden. Entweder wird in der genutzten „fetchmailrc“ Datei die korrekte Adresse des lokalen Empfängers eingetragen oder ***fetchmail*** wird mit der Option „-D domain.tld“ aufgerufen.

Letzteres kann auf openSUSE Systemen in

```
/etc/sysconfig/fetchmail
```

bzw. auf invis-Servern in

```
/var/cornaz/sysconfig/fetchmail
```

konfiguriert werden:

```
## Type:        string
## Default:     ""
#
# Any additional fetchmail options. See fetchmail(1) manual page for
# more information. If you want to use --mda option, it may be required
# to change FETCHMAIL_USER to root. Consult your MDA documentation for
# more.
#
FETCHMAIL_EXPERT_OPTIONS="-D invis-net.loc"
```

Eingetragen werden muss an dieser Stelle die lokale Domain des Servers.

Postfix - Mail-Annahme

Postfix spielt die zentrale Rolle im gesamten Mail-Setup.

Da wir es bezogen auf ein invis-Server-Setup „lediglich“ mit einem Mailserver in einem lokalen Netzwerk zu tun haben, werden die in der Grafik genannten Restrictions nicht konfiguriert. Unser Server bekommt emails entweder nur via ***fetchmail*** oder von lokalen Benutzern eingeliefert. In beiden Fällen ist das Einliefern von Mails ohne Beschränkung erlaubt.

Zunächst muss Postfix der eigene Zuständigkeitsbereich bekannt gemacht werden. Die zugehörigen Einstellungen werden in

```
/etc/postfix/main.cf
```

vorgenommen.

Die notwendigen Konfigurationen:

```
# INTERNET HOST AND DOMAIN NAMES
myhostname = server.invis-net.loc

mydomain = invis-net.loc
masquerade_domains = $mydomain

# SENDING MAIL
myorigin = $mydomain

# RECEIVING MAIL
inet_interfaces = $myhostname, localhost
....
mydestination = $myhostname, $mydomain, localhost.$mydomain, localhost
```

Dabei legt *myhostname* den lokal gültigen FQDN und *mydomain* die lokale Domain des Servers fest. Die Option *myorigin* definiert beim Versand von Mails die Absende-Domain, so nicht angegeben. Die Option *masquerade_domains* maskiert Subdomains oder Hostnamen in Email-Adressen mit der dahinter angegebenen Domain. D.h. Wird eine Mail von „host1.invis-net.loc“ versandt so würde Postfix im gezeigten Beispiel daraus schlicht „invis-net.loc“ machen. Emails versendende Hosts im lokalen Netz würden also hinter der lokalen Domain „versteckt“.

Dies kann als zusätzliche Sicherheitsmaßnahme verstanden werden, die dabei hilft, das lokale Mailrouting vor Fehlern zu schützen.

Über die Option *inet_interfaces* wird festgelegt, auf welchen Netzwerkschnittstellen Postfix überhaupt Mails annimmt. Die hier getroffene Einstellung erlaubt die Annahme nur auf der Loopback-Schnittstelle sowie allen Schnittstellen denen der eigene FQDN zugewiesen wird. In der Praxis wird hier häufig einfach der Wert „all“ angegeben, was bedeutet, das Postfix die Einlieferung von Mails auf allen Schnittstellen des Servers erlaubt. Dies schließt auch das externe, mit dem Internet verbundene Netzwerk-Device mit ein. In Kombination mit einer fehlerhaft konfigurierten Firewall, kann das

durchaus dazu führen, dass der Server von unerlaubter Seite als Mail-Relay missbraucht werden könnte.

Die Option *mydestination* legt die Ziele fest, für die Postfix Endpunkt der Email-Zustellung, also nicht weiterleitendes „Relay“ ist.

Nach der Annahme einer Mail, muss diese an AMaViS weitergegeben werden. Für die Einbindung von AMaViS als Viren- und Spam-Filter stehen mit „content_filter“ und „smtpd_filter_proxy“ zwei deutlich unterschiedliche Wege zur Verfügung. Während sich AMaViS mit der „smtpd_filter_proxy“ Methode direkt in die Einliefernde SMTP-Sitzung einklinkt und diese bis zum Abschluss der Prüfung offen hält, schließt sich die „content_filter“ Methode an die erfolgte Einlieferung an. Ist auf einem Internet-Mailserver Methode 1 die deutlich bessere, da hier „verseuchte“ gar nicht erst angenommen werden, so macht dieses Vorgehen auf einem invis-Server keinen Sinn, da die Mails hier vom lokalen Fetchmail-Dienst eingeliefert werden und somit eine Blockade nicht Zielführend ist.

Die Einbindung von AMaVis als Content-Filter findet an zwei Stellen im Postfix-Setup statt. In der Datei „main.cf“ erfolgt die eigentliche Integration des Filters durch Benennung eines „Transportweges“, welcher in

```
/etc/postfix/master.cf
```

konfiguriert wird. Zunächst die Datei „main.cf“:

```
## Virenscanner
content_filter=smtp-amavis:[127.0.0.1]:10024
```

Hier wird festgelegt, dass alle auf „normalem“ Wege eingehenden Mails über die Adresse 127.0.0.1 auf Port „10024“ dem Content-Filter zugeführt werden. Der eigentliche Transportweg wird hier mit „smtp-amavis“ benannt. Damit Postfix weiß, was es mit diesem Transport weg auf sich hat, muss dieser wie gesagt in der Datei „master.cf“ konfiguriert werden:

```
# Weitergabe an Amavis (betrifft nur content_filter, nicht
smtp_proxy_filter)
smtp-amavis     unix    -       -       y       -       2       smtp
    -o smtp_data_done_timeout=1200
    -o smtp_send_xforward_command=yes
    -o disable_dns_lookups=yes
    -o max_use=20
```

Wichtig hierbei sind vor allem die beiden Angaben in Zeile eins. Postfix wird mitgeteilt, dass die Weiterleitung per „smtp“ erfolgt und maximal zwei parallele Sitzungen gleichzeitig geöffnet werden dürfen. Die Anzahl der maximalen SMTP-Verbindungen kann in Abhängigkeit der lokalen Rechenleistung und dem Mail-Aufkommen variiert werden. In der Konfiguration des AMaViS-Dienstes befindet sich ein korrespondierender Parameter, der mindestens auf den gleichen Wert eingestellt sein muss.

Die nachfolgenden Optionen im Einzelnen:

- **smtp_data_done_timeout**: Timeout, nachdem die Weitergabe abgebrochen wird. Die Einstellung hier ist mit 1200 Sekunden doppelt so hoch wie der Vorgabewert, kann ja sein, dass AMaViS mal länger braucht. 😉

- **smtp_send_xforward_command**: Die Daten der einliefernden Sitzung wie Name, Adresse, Protokoll und HELO Name des original SMTP-Clients werden einfach an den Content-Filter weitergereicht.
- **disable_dns_lookups**: Es werden zur Ermittlung des Endpunkts keine DNS-Anfragen gestartet. Da ja ohnehin die IP-Adresse angegeben wurde ist das auch nicht notwendig.
- **max_use**: Postfix erlaubt maximal X eingehende Verbindungen für diesen Transportweg.

Von Bedeutung ist hier vor allem das „xforward“ Kommando, da die darüber an AMaViS übermittelten Informationen in die SPAM-Bewertung einfließen können.

AMaViS

Konfiguration

In der AMaViS-Konfiguration müssen keine besonderen Einstellungen für Annahme und Rückgabe der Mails vorgenommen werden, da die bisher besprochenen Transportwege den Standard-Vorgaben von AMaViS entsprechen.

Wichtig ist allerdings die Konfiguration des lokalen Host-Namens sowie der lokalen Domain in der Datei

```
/etc/amavisd.conf
```

Hinweis: *Bei der Datei handelt es sich um eine Perl-Datei, es ist also besonders auf korrekte Syntax zu achten.*

```
##invis-server.org -- Default-Settings
$mydomain = 'julgs-net.loc';      # (no useful default)

$myhostname = 'zentrale.julgs-net.loc';  # fqdn of this host, default by
uname(3)
```

Sollte der Weg der Rücklieferung vom Standard abweichen, müssen die folgenden Zeilen vom Kommentarzeichen befreit und entsprechend den Gegebenheiten angepasst werden:

```
#$forward_method = 'smtp:[127.0.0.1]:10025';  # where to forward checked
mail
#$notify_method = $forward_method;            # where to submit
notifications
```

Weiterhin ist darauf zu achten, dass die Anzahl der maximal gestarteten AMaViS-Prozessen gleich oder größer der für Postfix vorgenommenen Einstellung ist. Der im Beispiel genannte Wert war „2“:

```
$max_servers = 2;     # num of pre-forked children (2..30 is common), -m
```

Neue Konfigurationsdirektiven ab Version 2.6.x

In Versionen neuer 2.6.x ist Amavis in der Lage eingehende SMTP-Verbindungen zu cachen. Das reduziert auf hochfrequentierten Servern die CPU-Last. Auf nur gering belasteten Servern kann es jedoch zu Timeouts bei der Mail-Übergabe an Amavis kommen. Hier die originale Dokumentation dazu:

„ A current value of a global settings $smtp_connection_cache_enable controls whether a session will be retained after forwarding a message or not. Its default initial value is true. A global setting $smtp_connection_cache_on_demand controls whether amavisd is allowed to dynamically change the $smtp_connection_cache_enable setting according to its estimate of the message frequency. The heuristics is currently very simple: if time interval between a previous task completion by this child process and the arrival of a current message is 5 seconds or less, the $smtp_connection_cache_enable is turned on (which will affect the next message); if the interval is 15 seconds or more, it is turned off. The default value of the $smtp_connection_cache_on_demand is true, thus enabling the adaptive behaviour. On a busy server the connection caching can save some processing time. Savings are substantial if client-side TLS is enabled, otherwise just a few milliseconds are saved. On an idle server the feature may unnecessarily keep sessions to MTA open (until MTA times them out), so one can disable the feature by setting both controls to false (to 0 or undef). “

D.h. beide Werte sind auf gering belasteten Servern auf „0“ zu setzen um Postfix nicht zu verwirren. Das Verhalten von Postfix im Falle eines Timeouts ist merkwürdig. Unter Umständen scheint die eingehende SMTP-Sitzung noch gecached zu sein, d.h. Postfix hat noch kein finales Reject, versucht aber dennoch die Mail ein zweites Mal an Amavis zu übergeben. Gelingt der zweite Zustellversuch kommt es dazu, dass eine Mail einerseits ankommt aber der Absender dennoch nach Ablauf eines Timouts des ersten Versuchs eine Undelivery Meldung erhält. Konfusion perfekt!

Die Rückmail enthält folgende Fehlermeldung:

```
451 4.3.0 Error: queue file write error
```

Wird eine neue Amavisd-Installation mit einer alten Konfigurationsdatei betrieben, fehlen die zugehörigen Konfigurationsdirektiven und müssen ergänzt werden:

```
...
$smtp_connection_cache_on_demand = 0;
$smtp_connection_cache_enable = 0;
...
```

Beide Direktiven sind auf 0 zu setzen.

Postfix - Rücknahme und Weitergabe an Dovecot

Postfix nimmt von AMaViS geprüfte und nicht beanstandete Mails auf Port „10025“ wieder entgegen. Auch hierfür muss in

```
/etc/postfix/master.cf
```

ein Transportweg eingerichtet werden:

```
## Ruecktransport von amavis an Postfix
localhost:10025 inet    n       -       n       -       -       smtpd
```

```
  -o smtpd_tls_security_level=none
  -o content_filter=
  -o smtpd_proxy_filter=
  -o mynetworks=127.0.0.0/8
  -o smtpd_delay_reject=no
  -o smtpd_client_restrictions=permit_mynetworks,reject
  -o smtpd_helo_restrictions=
  -o smtpd_sender_restrictions=
  -o smtpd_recipient_restrictions=permit_mynetworks,reject
  -o smtpd_data_restrictions=reject_unauth_pipelining
  -o smtpd_end_of_data_restrictions=
  -o smtpd_restriction_classes=
  -o smtpd_error_sleep_time=0
  -o smtpd_soft_error_limit=1001
  -o smtpd_hard_error_limit=1000
  -o smtpd_client_connection_count_limit=0
  -o smtpd_client_connection_rate_limit=0
  -o
receive_override_options=no_header_body_checks,no_unknown_recipient_checks
```

In Zeile 1 wird zunächst festgelegt, dass der Endpunkt dieses Transportweges ein SMTP-Dienst ist. Eine Beschränkung der maximalen Anzahl der Prozesse ist hier nicht notwendig, da hier ohnehin nur das ankommen kann was zuvor den bereits auf zwei max. Sitzungen beschränkten Weg durch AMaViS genommen hat. (Zugegeben, so ganz richtig ist das nicht, da durchaus auch andere Programme auf dem Server Mails auf 127.0.0.1:10025 einliefern könnten.)

Die weiteren Optionen im Einzelnen:

- **smtp_tls_security_level**: Da die gesamte Kommunikation über die Loopback-Schnittstelle des Servers läuft ist eine Verschlüsselung obsolet.
- **content_filter** & **smtpd_proxy_filter**: Beiden Optionen ist kein Wert zugewiesen, was dazu führt, dass bei der Rücklieferung der Mails diese nicht in Form einer Endlosschleife wieder an AMaViS übergeben werden.
- **mynetworks**: Legt das Netzwerk fest auf dem dieser Transportweg zur Verfügung steht. Hier werden die im normalen Postfix-Setup genannten Netzwerke auf dass Loopback-Netz beschränkt. Benötigt wird diese Einstellung vor allem für die nachfolgende Zeile.
- **smtpd_recipient_restrictions**: Hiermit wird die Einlieferung von Mails auf den zuvor benannten Netzwerken ohne weitere Restriktionen erlaubt. Alle Einlieferungsversuche aus anderen Netzen jedoch unterbunden.

Achtung: *In einigen der jüngeren invis-Releases (7.0/7.1) habe ich darüber hinaus die Option „-o local_header_rewrite_clients=“ gesetzt. Dies wird auch an anderer Stelle so angegeben. Diese Option sorgt allerdings dafür, dass das zuvor vorgenommene Address-Rewriting (aus lokaler Mail-Adresse mache externe Adresse) beim Rückliefern der Mail wieder rückgängig gemacht wird. Bedeutet die Empfänger einer solchen Mail sehen als Absende-Adresse die lokale Email-Adresse des Absender, die aber im Internet-Mailverkehr keine Gültigkeit hat.*

Postfix ist zwar durchaus in der Lage Mails direkt in sogenannte „Maildirs“ einzuliefern, allerdings ist dieser direkte Weg mit allerlei Nachteilen verbunden, da dadurch einige Stärken von Dovecot ungenutzt bleiben. Besser ist es die Verteilung der Mails in die jeweiligen Postfächer Dovecots zu überlassen und die Übergabe an Dovecot via LMTP (Local Mail Transfer Protocol) zu erledigen.

Üblicherweise sind Maildirs in den Home-Verzeichnissen der Benutzer zu finden. Dies birgt einige Nachteile und Gefahren in sich. Die größte Gefahr dabei, geht wohl von den Nutzern selbst aus, die in der Lage sind diese Verzeichnisse einfach zu löschen. Dass ein Benutzer im „Aufräumrausch“ einfach mal Verzeichnisse löscht, mit denen er nichts anfangen kann, dürfte wohl keine Seltenheit sein.

Weiterhin verhindern Maildirs in den Home-Verzeichnissen der Benutzer die Anwendung von IMAP-ACLs also die Freigabe von Mail-Ordnern für andere Nutzer. Ursache hierfür sind die meist restriktiven Dateisystem-Zugriffsrechte bei Home-Verzeichnissen, die gesetzte IMAP-ACLs überlagern. D.h. die Maildirs müssen an einen anderen Ort verfrachtet werden und einem neutralen Benutzer gehören.

Üblicherweise werden für diesen Zweck sowohl ein Benutzer als auch eine Gruppe „vmail“ angelegt:

```
linux:~ # groupadd -g 399 vmail
linux:~ # useradd -c "Benutzerkonto fuer Dovecot Mailhandling" -s /bin/false
-g vmail vmail
```

Anschließend müssen die Rechte am Arbeits und Mail-Spool-Verzeichnis so angepasst werden, dass *vmail* darin schreiben darf:

```
linux:~ # chown vmail.mail /var/lib/dovecot
linux:~ #  chmod 0775 /var/lib/dovecot
linux:~ #  chown -R .vmail /var/spool/mail
```

Damit Postfix weiß, auf welchem Weg es Mails lokal zuzustellen hat, muss dies in Abhängigkeit zur Empfänger-Domain bekannt gemacht werden. Genutzt werden für diesen Zweck entsprechende Lookup-Tables. Hintergrund ist hier, dass Postfix über die Empfänger keine Kenntnis hat oder braucht. Es bekommt einfach nur gesagt, welche Domains lokal „relayed“ werden und wer sich um die Zustellung kümmert. Für beide Zwecke gibt es mit *relay_domains* und *transport_maps* eigene Lookup-Tables. Da in unserem Fall beide die Relay-Domains Table den gleichen Aufbau hat wie die normale Transport-Table, wird Sie an als zusätzliche Transport-Table eingebunden. Vorgenommen wird diese Konfiguration wieder in der Datei:

```
/etc/postfix/main.cf
```

```
# Relay Domains und Transportwege-Regelungs Tabelle bekannt machen
relay_domains = hash:/etc/postfix/relay_domains
transport_maps = hash:/etc/postfix/transport, $relay_domains
```

Die in der Konfiguration benannte Datei „relay_domains“ muss manuell angelegt und mit folgendem Inhalt versehen werden:

```
# for relaying domain
# domain.de OK
invis-net.loc   lmtp:unix:private/dovecot-lmpt
```

Darin enthält Spalte 1 die Domain und Spalte 2 den zugehörigen Transportweg.

Das Beispiel zeigt eine LMTP-Anbindung via UNIX-Socket. Die entsprechende Socket-Datei muss von Dovecot bereit gestellt werden. Alternativ wäre auch ein TCP/IP-Socket über die Localhost-Schnittstelle möglich.

```
# for relaying domain
# domain.de OK
invis-net.loc   lmtp:[127.0.0.1]
```

Die TCP/IP Methode ist etwas einfacher in der Konfiguration, da seitens Dovecot kein Socket definiert werden muss. UNIX-Sockets sind allerdings etwas performanter, da die übertragene Datenmenge durch Wegfall des TCP/IP bedingten Overheads geringer ist.

Letzte Etappe - Dovecot

Ab invis-Server Version 7.1-R3 kommt Dovecot in Version 2.1, inzwischen schon Version 2.2 zum Einsatz. Erwähnenswert ist das, da sich an dessen Konfiguration mit Einführung von V. 2.x einiges geändert hat. Einerseits betrifft dies Namen von Konfigurationsoptionen und andererseits den Aufbau der Konfiguration an sich. Unter openSUSE wurde die Konfiguration der Übersicht halber in mehrere Dateien gesplittet. Im Verzeichnis „/etc/dovecot“ ist nach wie vor eine (wenn auch stark geschrumpfte) Datei „dovecot.conf“ zu finden. Sie dient quasi als Hauptkonfigurationsdatei in die alle weiteren Dateien inkludiert werden. Ähnlich wie das bereits beim Apache-Webserver der Fall ist.

Alle weiteren Dateien sind logisch unterteilt im Verzeichnis „/etc/dovecot/conf.d“ zu finden.

Bei aller Freude über Peer Heinleins neues Dovecot Buch, ist doch festzustellen, dass die Aufgabenstellung die sich für die Dovecot-Konfiguration auf einem invis-Server stellt, mit dem Buch allein nicht zu bewältigen ist. Peer trennt im Prinzip zwischen einer Installation mit lokalen Benutzern und einer Hosting-Situation mit virtuellen Benutzern. Auf einem invis-Server haben wir es zwar mit lokalen Benutzern zu tun, aufgrund der Vorzüge, die das mit sich bringt, möchte ich sie aber lieber wie virtuelle Benutzer behandeln, da dies einige Vorteile mit sich bringt.

Definieren wir zunächst die Ziele:

- Zugriff auf die LDAP-Benutzerkonten via PAM
- Ordner-Freigaben über IMAP-ACLs
- Sieve Filter und sei es nur für Abwesenheitsbenachrichtigungen
- Quotas, wenn auch nur zum Warnen
- Lokale Email-Zustellung via LMTP
- SASL-Auth Schnittstelle für Postfix (wird im nächsten Abschnitt beschrieben)

Mailversand - SMTP-Auth mit Dovecot-SASL

…to be continued.

Autostart von Docker Containern mittels Systemd

Docker Container, die einen Dienst anbieten sollen natürlich beim Start eines Systems möglichst automatisch starten. Außer der Docker-eigenen „Restart=always“ Möglichkeit, die bisweilen ein wenig nervig sein kann, bietet sich die Verwendung von ***systemd*** an.

Leider ist die in der Docker-Dokumentation beschriebene Service-Unit Vorlage ein wenig zu simpel

gestrickt. Sie funktioniert nicht immer. Es kommt vor, dass es mit dieser Vorlage während des Systemstarts unzählige scheiternde Versuche gibt einen Container zu starten und ***systemd*** dann irgendwann aufgibt.

Daher hier eine Vorlage, die auch mit störrischen Containern funktioniert:

```
[Unit]
Description=BlaBlaBla Firebird-Server Container
Requires=docker.service
After=docker.service network.service

[Service]
Restart=on-failure
RestartSec=10
ExecStart=/usr/bin/docker run --volume=/srv/docker/blablafb/data:/data --
publish=3050:3050 --name=leberwurst blablablasoftware/db_server
ExecStop=/usr/bin/docker stop -t 2 leberwurst
ExecStopPost=/usr/bin/docker rm -f leberwurst

[Install]
WantedBy=multi-user.target
```

Dieser Vorlage unterscheidet sich an mehreren Punkten von der in der Docker-Doku gezeigten.

Zunächst wird für die Restart-Option der Parameter „on-failure“ statt „always“ genutzt. Dies verhindert unnötige Neustarts. Ich vermute, dass die Rückmeldungen von Docker nicht dem entsprechen, was ***systemd*** erwartet und daher mit „always“ Neustarts ausgelöst werden, die nicht notwendig sind.

Auch im Fehlerfall sollte zwischen den Neustarts eine Pause liegen, da der Docker-Container ggf. nicht so schnell agiert, wie ***systemd*** das erwartet. „RestartSec=10“ setzt diese Pause auf 10 Sekunden.

Da ich im Beispiel dem Container mit „–name=“ einen reproduzierbaren Namen gebe, scheitern zu schnelle Neustarts oft daran, dass bereits ein Container mit diesem Namen existiert (…auch wenn dieser nicht funktioniert). Mit „ExecStopPost=“ wird nach einem Stoppen des Containers dieser auch entfernt. Damit steht er einem Neustart unter gleichem Namen nicht im Weg.

Gespeichert werden die Container-Service-Unit Dateien unter:

```
/etc/systemd/system
```

Eine Veränderung dieser Dateien muss mit

```
linux:~ # systemctl daemon-reload
```

übernommen werden.

From:
https://wiki.invis-server.org/ - **invis-server.org**

Permanent link:
https://wiki.invis-server.org/doku.php?id=kb

Last update: **2020/10/20 10:47**