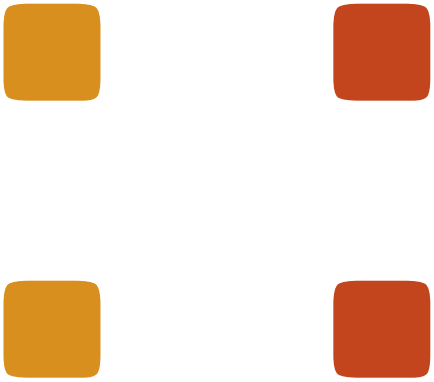

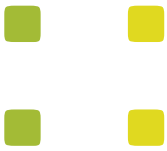

Reference manual

version 2.2.3

Y:A:D::I::F:A

Yet Another DNS Implementation For All

Contents

List of Figures

1 Introduction

YADIFA is a *name server* implementation developed by **EUR***id* **vzw/absl** (EURID), the registry for the *.eu* top-level domain name. EURID developed *YADIFA* to increase the robustness of the *.eu* name server infrastructure by adding a stable alternative to the other name server implementations in use.

In a nutshell, *YADIFA* :

- is an authoritative name server, in both a master and slave configuration
- is **RFC** compliant
- is portable across multiple Operating Systems including GNU/Linux, BSD and OSX
- is written from scratch in C. It is a clean implementation, which uses the OpenSSL library.
- supports *EDNS0*[57] (*EDNS0*)
- supports *Domain Name System Security Extensions*[49] (*DNSSEC*) with NSEC[50] (*NSEC*) and NSEC3[13] (*NSEC3*)
- has full and incremental zone transfer handling (*AXFR*[57] (*AXFR*) and *IXFR*[45] (*IXFR*)).
- *DNSSEC* signing service

The secondary design goals for *YADIFA* are to:

- be a caching name server
- be a validating name server
- have a backend which is Structured Query Language (SQL)-based[1]
- allow dynamic zone updates

[1] *YADIFA* will read zone from files and SQL-based backends

In future releases new features will be added, including:

- recursion
- caching
- validation
- split horizon
- plug-in system to integrate with EURID's proprietary systems
- dynamic provisioning of new domain names

1.1 Domain Name System

The *Domain Name System*[44] (*DNS*) is a system and network protocol used on the Internet. *DNS* is a globally distributed database with domain names, which can translate those domain names into IP addresses and vice versa. All Internet-connected systems (routers, switches, desktops, laptops, servers, etc.) use *DNS* to query *DNS* servers for a IP addresses.

DNS is used by most services on the Internet. Mail, which itself uses the SMTP-protocol, uses *DNS* to get information about where to send emails.

DNS is an hierarchical, distributed system (see figure 1.1). One *DNS* server cannot hold all the information.

If you want to surf to http://www.eurid.eu for example, your computer needs the IP address of *www.eurid.eu*.

It first asks to the *root* name servers which guide you to the *.eu* name servers, which in turn guides you to the EURID name servers, where you will get the IP address of *www.eurid.eu*.

1.1.1 Zones

The information about a domain name can be found in **zones**. In these **zones** you will not only find a website's IP address, eg. *www.eurid.eu*, or a mail server's IP address, but also the information that points you to a subsection of the **zone**.

To clarify:
To find the IP address of *www.eurid.eu*, you start your search at the *root* server. You are not given the website's IP address, but are pointed in the direction where you will be able to find the information. The *root* server points you to a subsection of its zone, it points you to the name server(s) of *.eu*. This we call a *delegation*. The **zone** information has a *Name Server*[44] (*NS*) resource record (*RR*) which contains the names of the *.eu* name servers. In the *.eu* zone information

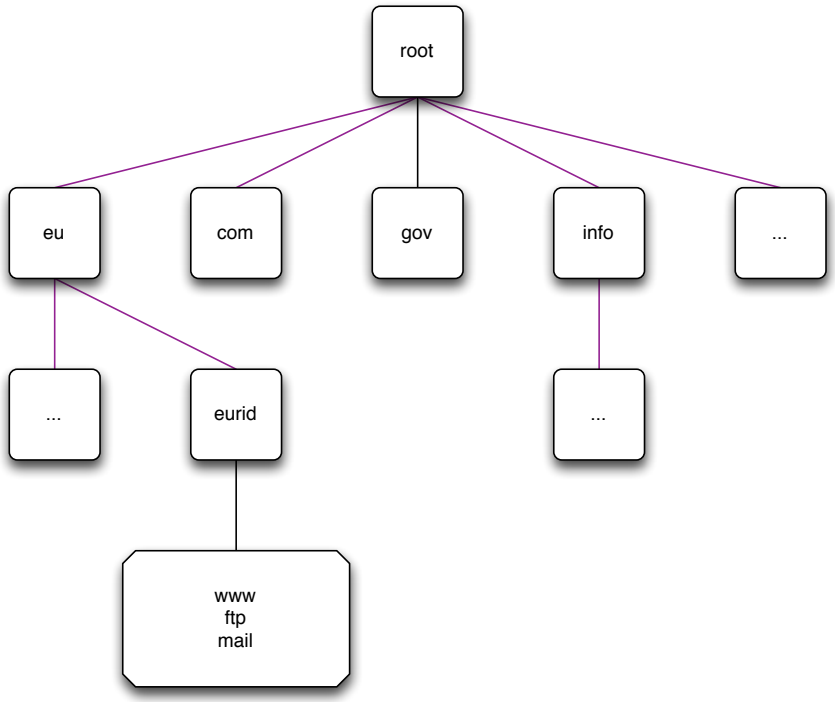

Figure 1.1: DNS hierarchy

you will still not find the IP address of the *www.eurid.eu* website, but you will find the **delegation** to the next domain name, *eurid.eu*. In the name servers of *eurid.eu* you will find the IP address of *www.eurid.eu*.

1.1.2 Authoritative name servers

Name servers with all the information for a particular zone are the *authoritative name servers* for that zone. When querying the information of a domain name with an **authoritative** name server, the name server will give not only the answer, but will also indicate that it is **authoritative** for the information it has provided, by sending an **Authoritative Answer** flag along with the result.

For redundancy purposes a zone does not have only one authoritative name server. Good practice is to have a second and/or third name server in a different sub network.

Primary name server

Only one name server has the original zone information. Most name servers have this kind of information in a text file, also known as a **zone file**. Which authoritative name server is the *primary name server* of a domain name can be found in the *Start Of Authority* (*SOA*) *RR*. This information can be obtained from any of the domain name's authoritative name server(s).

Sometimes a *primary name server* is called **master name server**.

Secondary name server

The **secondary name server** has the same information as the *primary name server*, but differs in that it does not have the original *zone file*. A **secondary name server** receives its initial information from a transfer of the *primary name server*. There are several techniques for getting this information.

Sometimes a *secondary name server* is called **slave name server**.

2 Resource Requirements

2.1 Hardware

2.1.1 CPU

The Central Processing Unit (CPU) must be able to handle 64-bit integers (natively or through the compiler). It has to run a memory model where the data pointer size must be equal to the code pointer size. Threading is also required.

2.1.2 Memory

One record takes about 135 bytes of memory. Enabling *DNSSEC* is more expensive and triples that value. At runtime, zone management and processing may require additional storage space, up to 150% of the zone file size.

2.2 Supported Operating Systems

Please find below a list of operating systems and architectures we support and which are known to work.

OS	x86_64	x86_32	ppc64	sparc
Redhat/CentOS 6+	Y	Y	Unknown	-
Fedora 19+	Y	Y	Unknown	-
Arch	Y	Y	-	-
Ubuntu	Y	Y	-	-
Debian	Y	Y	Y	Unknown
YellowDog	-	-	Y	-
FreeBSD	Y	Y	Unknown	Unknown
OpenBSD	Y	Y	Unknown	Unknown
OSX (10.9.4)	Y	-	-	-
Solaris 10	Y	Unknown	-	Y
Windows	Planned	Planned	-	-

SUPPORTED OSes

YADIFA has been compiled with GCC 4.9.1 64 bit on Solaris and OSX. Other Unix flavours (e.g. NetBSD) and Windows support are planned.

The architecture of *YADIFA* is very portable and will run on most flavours of GNU/Linux (e.g. OpenSUSE, Gentoo, Slackware,...) but these configurations are untested.

3 Installation

The current version of *YADIFA* is: 2.2.3

YADIFA is a collection of one daemon, *yadifad*; one client, *yadifa*; four libraries; four man pages, *yadifad.8*, *yadifa.8*, *yadifa.rc.5* and *yadifad.conf.5*; and example configuration files.

3.1 Server

- A daemon *yadifad*
- A man page *yadifad.8*
- A man page *yadifad.conf.5*
- A *yadifad.conf.example* file.

3.2 Client

- A remote access tool *yadifa* for the server *yadifad*
- A name server lookup tool *yadifa*
- A man page for *yadifa* *yadifa.8*
- A man page *yadifa.rc.5*.

3.3 Libraries

- *dnscore*
- *dnsdb*
- *dnszone*
- *dnslg*.

3.4 From Sources

Everything can be installed in a GNU fashion with *configure*, *make* and *make install*.

YADIFA is tested with:

- GCC 4.6 and GCC 6.1.1
- CLANG 3.8.0
- ICC 12.1.3.

If you want to compile *YADIFA* for a certain compiler you need to add the "CC" environmental variable:

There are compiler optimisation issues with GCC prior to version 4.6. *YADIFA* will compile and work with older GCC versions, provided that the code is compiled without any optimisation (use the flags -O0).

```shell
$ ./configure CC=gcc-4.6
```

or

```shell
$ ./configure CC=clang
```

or

```shell
$ ./configure CC=icc
```

3.4.1 Configure Options

You can configure *YADIFA* with several options, the most notable options available:

Functionality

OPTION	DESCRIPTION
–enable-shared	build shared libraries [default=no]
–enable-static	build static libraries [default=yes]
–disable-rrl	Disable DNS Response Rate Limiter
–enable-messages	Enable use messages instead of send (needed if you use more than one IP aliased on the same network interface)
–disable-master	Disable DNS master
–enable-ctrl	Enable remote control
–disable-nsid	Disable NSID support
–disable-acl	Disable ACL support
–disable-tsig	Disable TSIG support
–disable-dynupdate	Disable dynamic update support
–disable-rrsig-management	Disable RRSIG verification and generation for zones
–enable-non-aa-axfr-support	Enable Allows AXFR answer from master without AA bit set (Microsoft DNS)
–enable-lto	Enable LTO support, requires gold linker
–without-tools	build "build without the DNS tools"
–disable-zalloc	Disable zalloc memory system
–enable-log-thread-id	Enable write the thread id in each line of log
–enable-log-pid	Enable write the pid in each line of log
–enable-full-ascii7	Enable YADIFA will now accept ASCII7 characters in DNS names (not recommended)
–disable-ecdsa	Disable Elliptic Curve (ECDSA) support (i.e.: when the available OpenSSL does not supports it)

CONFIGURE OPTIONS

Location

OPTION	DESCRIPTION
–prefix=PREFIX	install architecture-independent files in PREFIX [/usr/local]
–exec-prefix=EPREFIX	install architecture-dependent files in EPREFIX [PREFIX]
–bindir=DIR	user executables [EPREFIX/bin]
–sbindir=DIR	system admin executables [EPREFIX/sbin]
–sysconfdir=DIR	read-only single-machine data [PREFIX/etc]
–localstatedir=DIR	modifiable single-machine data [PREFIX/var]
–libdir=DIR	object code libraries [EPREFIX/lib]
–includedir=DIR	C header files [PREFIX/include]
–datarootdir=DIR	read-only arch.-independent data root [PREFIX/share]
–mandir=DIR	man documentation [DATAROOTDIR/man]
–docdir=DIR	documentation root [DATAROOTDIR/doc/yadifa]

CONFIGURE OPTIONS

3.4.2 Server installation

When installing *YADIFA* in /opt/, the install_prefix needs to be set to /opt/

shell

```
$ install_prefix='/opt/'

$ tar zxvf yadifa-2.2.3-xxxx.tar.gz
$ cd yadifa-2.2.3-xxxx

$ ./configure --prefix=${install_prefix}/yadifa/
$ make
$ sudo make install
```

After the installation a tree structure with files will have been created:

```
${install_prefix}/bin/
${install_prefix}/etc/
${install_prefix}/include/dnscore/
${install_prefix}/include/dnsdb/
${install_prefix}/include/dnslg/
${install_prefix}/include/dnszone/
${install_prefix}/lib/
${install_prefix}/sbin/
${install_prefix}/share/man/man5/
```

```
${install_prefix}/share/man/man8/
${install_prefix}/share/doc/yadifa
${install_prefix}/var/log/
${install_prefix}/var/run/
${install_prefix}/var/zones/keys/
${install_prefix}/var/zones/masters/
${install_prefix}/var/zones/slaves/
${install_prefix}/var/zones/xfr/
```

The most important files are found in:

```
${install_prefix}/etc/yadifad.conf
${install_prefix}/bin/yadifa
${install_prefix}/sbin/yadifad
${install_prefix}/share/man/man5/yadifa.rc.5
${install_prefix}/share/man/man5/yadifad.conf.5
${install_prefix}/share/man/man8/yadifa.8
${install_prefix}/share/man/man8/yadifad.8
```

Depending on the manner of compilation you will find the libraries in:

```
${install_prefix}/lib/
```

and the include files in:

```
${install_prefix}/include/dnscore/
${install_prefix}/include/dnsdb/
${install_prefix}/include/dnslg/
${install_prefix}/include/dnszone/
```

3.5 From Packages

3.5.1 RHEL/CentOS/Fedora

YADIFA source and binary packages are available from EPEL (Extra Packages for Enterprise Linux),provided by Denis Fateyev. For the latest status, please check : Fedora Status Page

Preparation

We would like to refer you to the proper installation guide at https://fedoraproject.org/wiki/EPEL

- RHEL5/CentOS5 : yum install http://nl.mirror.eurid.eu/epel/5/i386/epel-release-5-4.noarch.rpm
- RHEL6/CentOS6 : yum install http://nl.mirror.eurid.eu/epel/6/i386/epel-release-6-8.noarch.rpm
- RHEL7/CentOS7 : yum install http://nl.mirror.eurid.eu/epel/7/x86_64/e/epel-release-7-2.noarch.rpm
- Fedora19+ : No special action required.

Installation

Once the repositories are setup, installation can be completed using the following command:

```shell
$ sudo yum install yadifa
```

3.5.2 Debian

Preparation

When using Debian STABLE, the package is not in the official stable repository and needs to be built manually.

```shell
$ sudo apt-get install build-essential \
                       dh-autoreconf \
                       dh-systemd \
                       unzip \
                       curl \
                       libssl-dev
$ curl -L https://github.com/asciiprod/yadifa/archive/11e1b33.zip -o master.zip
$ unzip master.zip
$ curl http://yadifa.eu/r/yadifa-2.2.2-6587.tar.gz -o yadifa_2.2.2.orig.tar.gz
$ cd yadifa-11e1b33*
$ dpkg-buildpackage -us -uc
$ cd -
```

The packages are now available as yadifa_<mainver>-<revision>_<architecture>.deb

Installation

```shell
$ sudo groupadd yadifa
$ sudo dpkg -i yadifa_2.2.2-2_*.deb
```

When using Debian sid, version 2.2.2 is available through the official repositories.

```shell
$ sudo apt-get install yadifa
```

3.5.3 Ubuntu

Preparation

The package is available through the official [universe] repository since Xenial Xerus (16.04 LTS)

```shell
$ sudo apt-get install yadifa
```

For older versions of Ubuntu, the package is not in the official repository and needs to be built manually.

Please follow the debian build procedure.

3.5.4 Arch Linux

YADIFA is available from AUR (Arch User Repository), provided by BlackIkeEagle.

Preparation

You are encouraged to read aur.archlinux.org for a full description on how to use AUR (Arch User Repository).

The package is available at **Yadifa AUR**

shell

```
$ curl https://aur.archlinux.org/packages/ya/yadifa/yadifa.tar.gz \
      -o yadifa.tar.gz
$ tar zxvf yadifa.tar.gz
$ cd yadifa
$ makepkg
```

Installation

Once the repositories are setup, installation can be completed using the following command:

shell

```
$ sudo pacman -U yadifa-2.1.6-1-x86_64.pkg.tar.xz
```

Or when you have installed pacaur, the preparation step can be skipped.

shell

```
$ sudo pacaur -S yadifa
```

3.5.5 Gentoo

Currently there is no emerge package available for Gentoo.

Please follow the source install option.

3.5.6 FreeBSD

YADIFA is available from FreeBSD ports

Installation

```shell
# cd /usr/ports/dns/yadifa && make install clean
# pkg install dns/yadifa
```

YADIFA is now installed in /usr/local

3.5.7 OpenBSD

Currently there are no packages or ports available for OpenBSD.

Please follow the source install option.

3.5.8 Solaris

There are no packages available for Solaris.

Please follow the source install option.

3.5.9 Mac OS X

Currently there is no Mac OS X package available.

Please use the source install.

4 Server Configuration

YADIFA is an authoritative name server only. Currently it does not have the functionalities to be a *caching name server*, a *validating name server* or a *forwarder*.

YADIFA can start up without prior configuration, and it just requires an empty configuration file. Of course with an empty configuration file it does not do much, but you can test certain functionalities. It will answer queries, but with no zones configured it will return a flag which indicates that the query has been refused (*REFUSED*). This flag will be explained later in the manual.

All logs will be will be sent to the standard output.

The *YADIFA* configuration file has thirteen sections:

Eight standard sections:

- *"main" section* (see on page 69) (<*main*>)
- *"zone" section* (see on page 74) (<*zone*>)
- *"key" section* (see on page 76) (<*key*>)
- *"acl" section* (see on page 77) (<*acl*>)
- *"channels" section* (see on page 79) (<*channels*>)
- *"loggers" section* (see on page 82) (<*loggers*>)
- *"nsid" section* (see on page 85) (<*nsid*>)
- *"rrl" section* (see on page 86) (<*rrl*>)

And five sections for DNSSEC-Policy (see on page 42) (DNSSEC-Policy) only:

- *"dnssec-policy" section* (see on page 87) (<*dnssec-policy*>)
- *"key-suite" section* (see on page 88) (<*key-suite*>)

- *"key-roll" section* (see on page 89) (<*key-roll*>)
- *"key-template" section* (see on page 90) (<*key-template*>)
- *"denial" section* (see on page 91) (<*denial*>)

Each section has its own set of configuration elements.

- <*main*> contains all the configuration parameters needed to start up *YADIFA*
- <*zone*> contains all the configuration parameters needed for the zones
- <*channels*> and <*loggers*> are needed to configure your log information
- <*key*> contains *TSIG*[32] (*TSIG*) information
- <*nsid*> contains the "DNS Name Server Identifier Option"
- <*rrl*> contains the "Response Rate Limiting in the Domain Name System".
- <*dnssec-policy*> (see chapter 8).

The configuration file also supports the use of **include**s. Included configuration files can itself contain **include** directives, with a maximum depth of 255. Relative path names will be treated as relative from the path of the configuration file where the **include** directive was defined.

configuration

```
<some_section>
 ...
</some_section>

include "../relative/to_this_file/include.conf" # with or without quotes
include include.conf                            # same directory as the
                                                # current file

<other_section>
 ...
</other_section>

include /absolute/path/to/file.conf             # absolute path
```

note

Included files are included in-line. This means the order is respected and later sections and configuration options overwrite previously defined options.

4.1 An authoritative name server

To allow *YADIFA* to answer queries for its domain names, you have to declare them to the *zone* section.

4.1.1 Primary name server

An example of a zone with domain name *somedomain.eu*.

configuration example

```
<zone>
    domain          somedomain.eu
    file            masters/somedomain.eu.
    type            master
</zone
```

Where:

- **domain** is the full qualified domain name
- **file** is the absolute or relative path of the zone file in text format
- **type** is the kind of name server *YADIFA* is for this zone. **type** can be:
  - Master
  - Slave.

In this example, *YADIFA* is configured as a *master*. This means that the original zone file is on this server and you need to edit the zone file on this server.

note

For a working example you can find the zone file on page 92.

4.1.2 Secondary name server

YADIFA is authoritative for the zone *somedomain.eu*, but does not have the original information. *YADIFA* needs to get the information from a *master* for this zone file.

configuration example

```
<zone>
    domain          somedomain.eu
    file            slaves/somedomain.eu.
    type            slave
    master          192.2.0.1
</zone>
```

In this example the **type** changes to *slave*. *YADIFA* needs to know where it can find the master zone file. This will be done with the additional configuration parameter **master**, where you can specify the IP address of the master name server for this domain name.

4.2 Signals

On a unix-like operating systems you can send a *signal* to a process, this is done with the *kill* command.

A few signals are implemented:

- **SIGTERM** will shutdown *YADIFA* properly
- **SIGINT** will shutdown *YADIFA* properly
- **SIGHUP** will reopen the log files and reload all updated zone files from disk. [1]
- **SIGUSR1** will save all zone files to disk. Zones files matching the zone in memory will not be overwritten.

For example:

shell

```
$ ps -ax | grep yadifad
67071   2  S+     0:03.47 ./yadifad
$ kill -HUP 67071
```

[1] only the zone files with an higher serial number on disk than in the database will be affected

5 Server Technical

For now there are three entry points to the database:

1. Zone File
2. *AXFR* and *IXFR*
3. *Dynamic Updates in the Domain Name System*[14] (*DNS UPDATE*).

All three use the same principles to accept a resource record:

- First-come, first-served
- Semantic errors will drop the relevant resource record
- Syntax errors will drop the relevant entity.

Dropping the relevant entity can mean several things. If a syntax error occurs in a *DNS UPDATE* just this packet will be dropped and not the relevant zone file. A syntactical error can be a typo, but for security reasons the entity will be dropped completely.

If a syntax error is not a typo, but something against the **RFCs**, only that resource record will be dropped.

5.1 Zone file reader

The zone file reader will check each resource record as a single entity. Inconsistencies are only checked once the whole zone has been loaded.

What are inconsistencies?

- The apex of a zone file

- Semantics of a resource record
- CNAME's alongside non-cname's
- Non-CNAME's alongside cname's
- Non-existing MACROS/DIRECTIVES (eg.typos in MACROS/DIRECTIVES).

5.1.1 Known types

For more information see section 13.3.

6 Client

YADIFA comes with one client:

1. yadifa

6.1 YADIFA

yadifa is the tool used to access the *yadifad* servers. *yadifa* can be used to configure a name server and control a name server.

yadifa communicates with the name server over a **Transmission Control Protocol**[47] (TCP) connection. This communication can be authenticated with *TSIG*'s. This *TSIG* can be given via the command line or a configuration file.

If you want to have control support in *YADIFA* you need to enable this function before compiling the sources.

shell
```
$ ./configure --enable-ctrl
```

After the *configure*, you can do the normal *make* and *make install*.

shell
```
$ make
$ make install
```

note

You also need to add allow-control in the <*main*> of *yadifad.conf*(12.3.1).

6.1.1 Commands

TYPES	ARGUMENTS
CFGRELOAD	somedomain.eu
FREEZE	somedomain.eu
LOGREOPEN	
QUERYLOG	–disable , –enable
RELOAD	somedomain.eu
SHUTDOWN	
SYNC	somedomain.eu
UNFREEZE	somedomain.eu
ZONECFGRELOAD	somedomain.eu

COMMANDS

cfgreload

This command will reload all keys, and the zones configurations and the zones. The port can be optionally supplied.

shell example

```
$ yadifa -s "192.0.2.1 port 53" -t CFGRELOAD
```

Gives as result in verbose mode:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 29457
;; flags: qr QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                  CTRL    CFGRELOAD

;; Query time: 4 msec
;; WHEN: Wed May  6 14:24:15 2015
;; MSG SIZE rcvd: 17
```

freeze

This command suspends updates to a zone. No further modifcations (*DNS UPDATE*) can be made.

shell example

```
$ yadifa -s 192.0.2.1 -t FREEZE -q somedomain.eu
```

Gives as a result in the verbose mode:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 3507
;; flags: qr QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                  CTRL    FREEZE

;; ANSWER SECTION:
.                           0       CTRL    FREEZE  \# 15 A037F6D65646F6D61696
E620565700

;; Query time: 0 msec
;; WHEN: Mon Sep 29 14:55:20 2014
;; MSG SIZE rcvd: 43
```

logreopen

This command reopens all log files.

shell example

```
$ yadifa -s 192.0.2.1 -t LOGREOPEN
```

Gives as a result in the verbose mode:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 12803
;; flags: qr QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                   CTRL    LOGREOPEN

;; Query time: 570 msec
;; WHEN: Wed May  6 14:29:57 2015
;; MSG SIZE rcvd: 17
```

querylog

This command enables or disables query logs.

shell example

```
$ yadifa -s 192.0.2.1 -t QUERYLOG --enable
```

Gives as a result in the verbose mode:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 10572
;; flags: qr QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                  CTRL    QUERYLOG

;; ANSWER SECTION:
.                          0       CTRL    QUERYLOG  \# 1 31

;; Query time: 1 msec
;; WHEN: Wed May  6 14:30:42 2015
;; MSG SIZE rcvd: 29
```

reload

This command reloads the zone file from disk. If no parameter is given, *'.'* will be used as domain name.

shell example

```
$ yadifa -s 192.0.2.1 -t RELOAD -q somedomain.eu
```

Gives as a result in the verbose mode:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: ?, status: NOERROR, id: 1750
;; flags: qr QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                  CTRL    RELOAD

;; ANSWER SECTION:
.                          0       CTRL    RELOAD  somedomain.eu
E620565700

;; Query time: 1 msec
;; WHEN: Mon Sep 29 15:01:34 2014
;; MSG SIZE rcvd: 43
```

shutdown

This command shuts down the server.

shell example

```
$ yadifa -s 192.0.2.1 -t SHUTDOWN
```

Gives as a result in the verbose mode:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOTAUTH, id: 57004
;; flags: qr QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                  CTRL    SHUTDOWN

;; Query time: 0 msec
;; WHEN: Mon Sep 29 14:46:50 2014
;; MSG SIZE rcvd: 17
```

sync

This command writes the zone to disk and optionally removes the journal. If no zone is specified, all zones are implied. The extra [–clean] option will remove the journal.

shell example

```
$ yadifa -s 192.0.2.1 -t SYNC -q somedomain.eu --clean
```

Gives as a result in the verbose mode:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 46355
;; flags: qr QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                  CTRL    SYNC

;; ANSWER SECTION:
.                          0       CTRL    SYNC    \# 5 A037F6D65646F6D61696
E620565700

;; Query time: 2 msec
;; WHEN: Wed May  6 14:35:27 2015
;; MSG SIZE rcvd: 33
```

unfreeze

This command enables updates to a zone. Modifications (*DNS UPDATE*) can be done again.

shell example

```
$ yadifa -s 192.0.2.1 -t UNFREEZE -q somedomain.eu
```

Gives as a result in the verbose mode:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 26357
;; flags: qr QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                  CTRL    UNFREEZE

;; ANSWER SECTION:
.                          0       CTRL    UNFREEZE  \# 15 A037F6D65646F6D61696
E620565700

;; Query time: 0 msec
;; WHEN: Mon Sep 29 14:56:49 2014
;; MSG SIZE rcvd: 43
```

zonecfgreload

This command rereads the zone config and reloads the zone file from disk.

shell

```
$ yadifa -s 192.0.2.1 -t ZONECFGRELOAD -q somedomain.eu
```

Gives as a result:

shell output

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: CTRL, status: NOERROR, id: 49879
;; flags: qr QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                  CTRL    ZONECFGRELOAD

;; ANSWER SECTION:
.                          0       CTRL    ZONECFGRELOAD  \# 15 A037F6D65646F6D6
1696E620565700

;; Query time: 1 msec
;; WHEN: Tue Sep 30 09:39:23 2014
;; MSG SIZE rcvd: 43
```

7 Domain Name System Security Extensions (DNSSEC)

7.1 Introduction

The *DNS* provides responses without validating their source. This means that it is vulnerable to the insertion of invalid or malicious information, a flaw discovered by Dan Kaminsky in 2008.

This technical report documents the various components of the long-term solution to this kind of cache-poisoning attack: *DNSSEC*.

7.2 DNSSEC overview

In a nutshell, *DNSSEC* adds signatures to regular *DNS* responses in the form of Resource Record Signature[50] (*RRSIG*). A signature covers a resource record set. A resource record set properly signed by a trusted source can be accepted as valid. Many signatures can cover the same resource record set.

The RRSIG resource record is consistent in a hash[1] of the covered resource record set along with the validity period and other relevant information, signed with the private part of the owner's key pair [2].

To be able to verify whether the response is legitimate, the receiver of a signed response should verify that each resource record set is verified by at least one of the signatures that covers it.

If this comparison shows no differences, the receiver is sure of two things:

- Integrity - the response has not been modified
- Authenticity - the response comes from the expected source

[1] A hash of a sequence of characters is the result of a one-way transformation of that sequence into a much smaller, fixed-length sequence by applying a certain mathematical formula. The slightest change of the original sequence changes the resulting hash. Thus, after transmission of the characters, one can detect changes to a sequence by comparing its current hash with the original.

[2] Public/private key encryption is well-known. A message is signed with the private part of a key pair (kept secret). The resulting signed message can only be verified using the public part of the key pair (shared with everybody).

(the only one to possess the private part of the key pair).

Note that the response itself is not encrypted. *DNSSEC* adds *RRSIG* records to responses, but the records that hold the data remain unaltered. In this way, *DNSSEC* is backwards compatible as non-*DNSSEC*-aware name servers can and should ignore unknown data and continue to function as expected.

The challenge in this scenario is to get the public part of the key pair to the users who need it for verification in a secure way.

The public parts of key pairs are available via the *DNS* as they are published as Domain Name System KEY[50] (*DNSKEY*) resource records. When querying for *DNSKEY* records, the response to a query also holds a signature for the *DNSKEY* record. But the question remains, should the receiver simply accept that the data is authentic and use it?

The answer is no. To verify the signature of a *DNSKEY* record, the user must consult the parent of the domain name. For domain names, such as eurid.eu, the parent is the Top Level Domain (*TLD*). For a *TLD*, the parent is the root domain. To enable users to obtain the public part of a signed domain name in a secure way, a hash of the public key is put in the parent zone as a Delegation Signer[50] (*DS*) resource record.

The parent zone signs the *DS* resource record with its keys, authentifying the delegation in the process. In the case of eurid.eu, a hash of the public key (*DS*) is put in the .eu zone where it is signed with the private key of .eu. For the .eu zone itself, a hash of the *.eu* public key (*DS*) is put in the root zone, where it is signed with the private key of the root zone.

This means that the receiver can obtain the public part of a key pair by querying for its hash in the parent zone, and, verify its signature with the public part of that parent-zone's key pair. This process only takes us up one level in the *DNS* hierarchy.

There the question repeats itself: how can the receiver trust the signature from that parent zone file? The answer lies in applying the same procedure: retrieving the public part of its key, the hash from its parent and the hash's signature.

But ultimately, some trust must be built in.

Herein lies the importance of having a signed Internet root zone, because receivers that verify signatures only need to trust the public key of the root zone. This is the only public key necessary and it can be obtained outside the *DNS*. It is available for download in several different formats together with a signature file at: http://data.iana.org/root-anchors/. Before the root zone was signed on 15 July 2010, administrators had to manually configure and maintain public key information from different branches in the *DNS* tree.

Now that the root zone is signed, one can imagine how much effort *TLD* operators are putting into enabling *DNSSEC* on the domains they serve. Only a complete chain of trust allows the secure authentification of a domain name.

7.3 Types of key pairs

Two types of keys are used in *DNSSEC*:

- The Key Signing Key[50] (*KSK*) - used only to sign the hash of *DNSKEY* information
- The Zone Signing Key[50] (*ZSK*) - used to sign the hashes of all resource records (A, NS, MX, etc).

The more signatures generated with a particular key pair, the greater the chance of a successful crypto-attack, in other words deducing the private part of a key pair by using the public part and the available signatures. To prevent the signing of false information, key pairs should not be used indefinitely. Every so often, new key pairs should be generated and used to resign the zone. The frequency of key generation depends on the strength of the algorithm, key length and how often a key is used.

Because strong algorithms and long keys require more resources, such as more CPU, the practice is to use a weaker key pair, the ZSK, for all signatures but to change it regularly. Validity of these signatures should be three to six months at most. A stronger key pair, the KSK, is only used to sign the public key information. The KSK is changed less frequently, every one to two years. Only a hash of the KSK appears in the root zone (as the DS record). Since this key is changed, or rolled over, less often, interaction with the parent is less frequent.

7.4 Algorithms

Several algorithms for calculating hashes and signatures have been defined. Specific name server implementations or versions may not support all of the algorithms mentioned in the following summary:

RSASHA1 (algorithm number 5) is declared mandatory by **RFC 4034**[50]. RSASHA1-NSEC3 - SHA1 (algorithm number 7) is defined by **RFC 5155**[13]. It is essentially the same algorithm as RSASHA1, although the Next SECure records are NSEC3. The stronger algorithms, RSASHA256 (algorithm number 8) and RSASHA512 (algorithm number 10) are both defined by **RFC 5702**[35].

The use of these latter algorithms is recommended, as attacks against SHA1 (used in algorithms 5 and 7) are increasing. Bear in mind that the newer algorithms, numbers 8 and 10, may not be available in older DNS server implementations and, as verifying DNS name servers that do not recognise an algorithm will treat the data as unsigned. It is unclear at the time of writing whether end users will actually benefit from these stronger algorithms.

8 DNSSEC Policies

8.1 Introduction

The DNS infractructure is an integral and critical part of the Internet. With that said, the introduction of *DNSSEC* did not make life easier for the hostmaster. Generation of *KSK*'s and *ZSK*'s, in addition to signing the zone using 'salt' and its iterations cause further complexity. To ensure that the keys will not be compromised, new keys must be generated continuously, at regular intervals, in a process called a 'key roll over'. When a key-roll over occurs, it is critical to not lose the integrity of the zone information. At no moment in time is it acceptable to have the zone unsigned or the keys, *KSK* and *ZSK*, outdated.

Due to these complex manipulations, especially on large amounts of zones in a portfolio, there is a need for an overall mechanism to facilitate *DNSSEC* enabled zones. Thanks to DNSSEC-policies the administrative overhead and complexity for *DNSSEC* enabled zones can be reduced significantly by generating and activating the keys automatically and maintaining the validity of the signatures.

8.2 What is needed for DNSSEC?

To implement DNSSEC, the following items are required:

- Keys for signing
- A signed zone
- A delegated zone

8.2.1 Keys for signing

In DNSSEC, there are two different types of keys for signing the zone. The *KSK* and *ZSK*. The only difference in both keys is the use.

The *KSK* (Key Signing Key) is used to sign the *DNSKEY* resource record set only and has the Secure Entry Point[50] (*SEP*) bit set. The *ZSK* (Zone Signing Key) is used to sign each resource record set of the zone. It is recommended to use a *KSK* in addition to a *ZSK*. The keysize *KSK* should be larger, resulting in stronger cryptography and therefore can be rolled-over less often.

Each key consists of two parts: one private the other public.

Private Key

This key is used for signing all the resource record sets. The signatures are stored in the *RRSIG* records and are only valid for a limited amount of time.

The current, most common format used to store a private key is depicted below:

```
Private-key-format: v1.3
Algorithm: 8 (RSASHA256)
Modulus: ...
PublicExponent: AQAB
PrivateExponent: ...
Prime1: ...
Prime2: ...
Exponent1: ...
Exponent2: ...
Coefficient: ...
Created: <create-date>
Publish: <publish-date>
Activate: <activate-date>
Inactive: <inactive-date>
Delete: <delete-date>
```

The fields; Created, Publish, Activate, Inactivate and Delete; indicate when the key must be used and when it must be removed from the zone.

- Created: Date the key was created.
- Publish: Date the public part of the key is published in the zone.
- Activate: Date the key should start signing the resource record sets.
- Inactivate: Date the key should stop signing the resource record sets.
- Delete: Date the public part of the key is removed from the zone.

Public Key

The public (part of the) key is used to verify the signatures generated by the private (part of the) key. The public key is published in the zone as the *DNSKEY*. The only difference between the *KSK* and *ZSK* is the presence of the *SEP* bit, resulting in 257 flags for *KSK* instead of 256 for a *ZSK*.

```
somedomain.eu. IN DNSKEY 257 3 8 AwE...
```

DS

The *DS* (Delegated Signer) record is the cryptographic glue between the parent and delegated zone. This record needs to be published in the parent zone and needs to correspond with *DNSKEY* in the delegated zone.

```
somedomain.eu.      86400   IN  DS  <keytag> 8 2 <hash_of_key>
```

8.2.2 Signed zone

A zone is signed when all the resource record sets are signed by a valid *ZSK*. To be valid, the *ZSK* itself needs to be published as a *DNSKEY* record and is to be signed by a *KSK*, which itself must also be published as a *DNSKEY*. The *KSK* must have a corresponding *DS* record in the parent zone and must in turn be signed by the parent's *ZSK*.

Depending on your preferences and/or requirements, a choice between *NSEC* and *NSEC3* must be made to prove the Denial of Existence.

Signatures

Signatures are generated by the private key and stored in the zone as *RRSIG* records.

```
somedomain.eu.      86400   IN RRSIG   <type_covered> 8 2 86400 (
                              <end_date> <begin_date> <key_tag> <signer>
                              <signature> )
```

Denial of Existence

DNSSEC requires a cryptographic proof of non-existence. The zone is sorted by the labels and *NSEC* or *NSEC3* records are generated representing the gaps between two subsequent labels. When a non-existing record is requested, the *NSEC* or *NSEC3* record is returned in between the requested record should have been found. The *NSEC* or *NSEC3* resource records are signed by an *RRSIG*.

For *NSEC*, the non-existence of *somedomain.eu.* would result in a reply similar to:

```
eu.                  7200    IN  NSEC    0.eu. NS SOA TXT RRSIG NSEC DNSKEY
somedicprod.eu.      7200    IN  NSEC    somedreams.eu. NS RRSIG NSEC
```

When using *NSEC3*, the mechanism is similar to *NSEC*, but all the records are hashed before being sorted. The hashing algorithm, the salt and the number of times it should be hashed are stored in an NSEC3PARAM[13] (*NSEC3PARAM*) record and are copied in each *NSEC3* resource record. In *NSEC3* there is an option to enable the Opt-Out[6] (*Opt-Out*). When this flag is set, only the zones for which there is a secure delegation will be considered for generating the *NSEC3* records. Non-secure delegations will be treated as non-existent and will reduce the number of *NSEC3* records being created significantly.

```
QBQ65Q6097OCPPROEUCQNSC1FHEO73UA.eu. 600 IN NSEC3 1 1 1 5CA1AB1E (
                 QBQ6OCGMT2JNIJ4JNF2CCRFI4CE4NUEO
                 NS SOA RRSIG DNSKEY NSEC3PARAM )
BKP4A7B3BOFKDVMPFABNCJO46PB2911A.eu. 600 IN NSEC3 1 1 1 5CA1AB1E (
                 BKPDVHUHA3S2PVTPI58DP5I5SABJUIM4
                 NS DS RRSIG )
4EIAT7URLC7FMN9AGIJ231E2S7L62TGO.eu. 600 IN NSEC3 1 1 1 5CA1AB1E (
                 4EIOQGMMDBOBP76VHHBDNVEN2UUNABGK
                 NS DS RRSIG )
```

8.2.3 Delegated zone

For *DNSSEC* to work, the whole chain up to the root must support *DNSSEC*. If the parent zone does not support *DNSSEC*, the chain cannot be verified and will not work.

8.3 What is needed for yadifa?

As there are a number of parameters to define, the components of *DNSSEC* policies span the following sections:

- *<zone>*
- *<dnssec-policy>*
- *<denial>*
- *<key-suite>*
- *<key-template>*
- *<key-roll>*

8.3.1 Zone

Any zone can be handled by *DNSSEC* policies.

If a zone is activated to handle *DNSSEC* by DNSSEC-policy, the keyword **dnssec-policy** with an associated **id** must be added.

configuration example of *<zone>* with dnssec-policy

```
<zone>
    domain          somedomain.eu
    file            masters/somedomain.eu.
    type            master

    dnssec-policy   "dp-1"
</zone>
```

8.3.2 DNSSEC-Policy

A DNSSEC-Policy configured zone needs *<dnssec-policy>* which has several keywords:

- **id**
- **denial**
- **key-suite**

configuration example of <*dnssec-policy*> with *NSEC3*

```
<dnssec-policy>
    # name of the 'dnssec-policy'
    id                      "dp-1"

    denial                  "nsec3-denial"

    # at least one: key-descriptor "name"
    # they define KSK & ZSK keys

    key-suite               "zsk-1024"
    key-suite               "ksk-2048"
</dnssec-policy>
```

At least one <*key-suite*> must be configured. It is also recommended to have one *KSK* and one *ZSK*. *YADIFA* will only read the first four **key-suite**s.

The argument of **key-suite** is a string that must be unique per section type. It is possible, however, to configure several different sections with the same name (id). For example, in one configuration it is possible to have a <*denial*> and a <*key-suite*> with the same "id".

If <*dnssec-policy*> contains two or more **key-suite**s that contain the same content, only one <*key-suite*> will be applied.

note

Clarifying the same content:

If two <*key-suite*> have the same definition about keys in addition to the same time schedule regardless of their names (ids), only one will be applied while the other is silently ignored.

8.3.3 Denial

The <*denial*> section contains several keywords:

- **id**
- **salt**[1]
- **salt-length**[1]
- **iterations**

[1] mutually exclusive, if both are defined, the system will refuse to start due to a parsing error

- **optout**

The zone can be signed with *NSEC* or *NSEC3*. If *NSEC3* has been chosen, salt will still need to be used for the *NSEC3PARAM* and the amount of iterations of this salt. In addition, the digest algorithm is also needed and is fixed to SHA1. This cannot be changed.

The choice between *NSEC* or *NSEC3* is done in the <*dnssec-policy*>.

Here are two examples:

- An example with the use of *NSEC*

configuration example of <*dnssec-policy*> with *NSEC*

```
<dnssec-policy>
    id              "dp-1"

    denial          "nsec"
    ...
    ...
</dnssec-policy>
```

- An example with the use of *NSEC3*

configuration example of <*dnssec-policy*> with *NSEC3*

```
<dnssec-policy>
    id              "dp-2"

    denial          "nsec3"
    ...
    ...
</dnssec-policy>
```

With the latter, "dp-2", there is still a need for <*denial*>. In <*denial*> you need to add a "salt" which can be blank. The algorithm used for the hashing of the *NSEC3 RR* is always SHA1 and cannot be changed. The parameters that can be set are: "**iterations**", which is the amount of iteration done; the salt which can be set with the mutually exclusive: "**salt**" or "**salt-length**"; and "**optout**" to enable or disable the opt-out feature of *NSEC3*. When the opt-out feature is enabled, *RRSIG*s for insecure delegations are not generated, resulting in smaller zones while maintaining the security for secure delegations.

salt is used as keyword with argument a string. This string is *BASE16*[36] (*BASE16*) and is the actual salt. The keyword **salt-length** will generate a random string with the length provided as argument.

configuration example of <*denial*> with keyword salt

```
<denial>
    id              "nsec3"

    salt            "BA53BA11"
#   salt-length     4
    iterations      5
    optout          off
</denial>
```

note

Default value of salt-length's arguments is "0". There is no salt if salt-length is "0".

8.3.4 Key Suite

A zone file can have several keys.

Preferably a zone file is configured with two keys:

- *KSK*
- *ZSK*

Configuration of the key is done in <*key-suite*>. The section has three keywords:

- **id**
- **key-template**
- **key-roll**.

key-template has the definition of the key and **key-roll** is the time schedule of the key.

configuration example of <*key-suite*>

```
<key-suite>
    id              "ksk-2048"

    key-template    "ksk-2048"
    key-roll        "key-roll-ksk-2048"
</key-suite>
```

note

A zone with only a *ZSK* is acceptable, but a zone with only a *KSK* is not.

8.3.5 Key Template

There are two kinds of keys:

- *KSK*

  configuration example of <*key-template*> with a *KSK*

```
<key-template>
    id              "ksk-2048"

    ksk             true
    algorithm       8
    size            2048
</key-template>
```

- *ZSK*.

  configuration example of <*key-template*> with a *ZSK*

```
<key-template>
    id              "zsk-1024"

    ksk             false
    algorithm       8
    size            1024
</key-template>
```

The arguments of **algorithm** and **size** keywords are referenced in the configuration reference chapter (12.3.12).

note

The key-suite is configured in <*dnssec-policy*>.

configuration example with of <*dnssec-policy*> with *NSEC3*

```
<dnssec-policy>
    id              "dp-2"
    denial          "nsec3-denial"
    key-suite       "ksk-2048"
</dnssec-policy>
```

8.3.6 Key-roll

A *DNSSEC* key has a life-span. It starts with creating (generating) the key and ends with removing the key from the zone file.

A time schedule has several phases:

- Generate a key
- Publish a key in a zone
- Activate a key
- Inactive a key
- Remove a key from the zone

The mechanism for changing one key with another is called a key-roll over. Key-roll overs follow the time schedule of a key. There are two kinds of 'key-roll' mechanism:

- Relative
- Diary

Key-roll mechanism 'relative' style

configuration example of <*key-roll*> with relative mechanism

```
<key-roll>
    id                  "key-roll-zsk-1024"

    create              +30d
    publish             +2h
    activate            +7200 # 2 hours (in seconds)
    inactive            +31d
    delete              +7d

</key-roll>
```

The <*key-roll*> with the relative mechanism has an **id** and time phases.

The time phase keywords are:

- **create**
- **publish**
- **activate**
- **inactive**
- **delete**.

One time phase has a keyword with 2 arguments. The first argument is a time period with a resolution in seconds. The second argument is the dependency of a time phase with a previous one.

For example, **publish** will be done 2 hours after the **generate** time phase. The activate time phase will be done another 2 hours later after the **publish** time phase.

note

The first argument always starts with a plus-sign.

note

The resolution of key rolls are in minutes. The seconds will be rounded up to the minute.

If the second argument is not given default values will be used. (see section 12.3.11)

Key-roll mechanism 'diary' style

configuration example of <*key-roll*> with diary mechanism

```
<key-roll>
    id                  "key-roll-ksk-2048"

#   command         minutes   hours     day       month     day-week  week
    create          0         8         4         2         *         *
    publish         0         12        4         2         *         *
    activate        0         12        14        2         *         *
    inactive        0         8         4         3         *         *
    delete          0         12        11        3         *         *
</key-roll>
```

The <*key-roll*> with the diary mechanism has an **id** and time phases.

These "time phases keywords" are the same as those in the relative mechanism.

One time phase has one keyword with 6 arguments. The first argument is the minutes of the hour, the second is the hours of the day. The third argument is the day of the week, and the fourth is the month of the year. The fifth argument can be used to specify a day in a week (e.g. Wed for Wednesday). The last argument is the week number in the month.

note

A mix of relative mechanism and diary mechanism styles in one <*key-roll*> is not allowed.

See section 12.3.11 for further explaination.

9 DNS Name Server Identifier (NSID)

9.1 Introduction

The *DNS* infrastructure is an integral and critical part of the Internet and the robustness of this system has constantly been improved since it was first used. The increased robustness has lead to more complex setups where mechanisms like *DNS* anycast, name server pools and IP failovers allow different name servers to be available from a single IP address. These complex setups can make it very difficult to identify individual name servers. To identify different name servers, one could query for a specific record which is unique to each of the name servers. However, this method will not work for generic queries which comprise the bulk of all requests. DNS Name Server Identifier[10] (*NSID*) provides a solution by including a unique identifier within any *DNS* response. This feature is an extension of the *DNS* protocol. To allow backward compatibility, a name server that has the *NSID* extension will only send an *NSID* when it is explicitly asked for. The information, in response to the *NSID* option in the query, can be found in the EDNS OPT pseudo-*RR* in the response.

9.2 NSID payload

The *NSID* is a sequence of up to 512 arbitrary bytes set by the administrator. When queried, the byte sequence is usually represented as an hexadecimal string followed by its corresponding ASCII chars, if possible.

The syntax and semantics of the content of the *NSID* option are deliberately left outside the scope of this specification.

Examples of *NSID*:

- It could be the “real” name of the specific name server within the name server pool.
- It could be the “real” IP address (IPv4 or IPv6) of the name server within the name server pool
- It could be a pseudo-random number generated in a predictable fashion somehow using the server’s IP address or name as a seed value

- It could be a probabilistically unique identifier initially derived from a random number generator then preserved across reboots of the name server
- It could be a dynamically generated identifier so that only the name server operator could tell whether or not any two queries had been answered by the same server
- It could be a blob of signed data, with a corresponding key which might (or might not) be available via *DNS* lookups.

10 DNS Response Rate Limiting (RRL)

10.1 Introduction

A typical Distributed Denial of Service (DDoS) attack relies on a great number of hosts to send many requests simultaneously to disrupt a service. *DNS* is at the core of the Internet and when this service is disrupted, many other services are disrupted as well as collateral damage. Therefore many *DNS* service providers have made major investments in good connectivity to mitigate attacks directed at their infrastructure. A *DNS* amplification attack is a special form of DDoS which takes advantage of the stateless nature of *DNS* queries to create forged *DNS* requests. Answers to these requests are sent to the actual target of the attack. The *DNS* protocol has been designed with efficiency in mind. Therefore a typical request requires a minimal amount of bandwidth to the name server, but can trigger a huge response which is typically many times larger than the original request. These huge responses allow attackers to hedge their disposable bandwidth with the bandwidth available at some DNS servers by making them unwilling participants in this special form of DDoS.

10.2 What is it?

The DNS Response Rate Limiting is an algorithm that helps mitigating *DNS* amplification attacks. The name servers have no way of knowing whether any particular *DNS* query is real or malicious, but it can detect patterns and clusters of queries when they are abused at high volumes and can so reduce the rate at which name servers respond to high volumes of malicious queries.

10.3 The problem

Any internet protocol based on **User Datagram Protocol**[46] (UDP) is suitable for use in a Denial of Service (DDoS) attack, but *DNS* is especially well suited for such malevolence. There are several reasons:

- Reflected/Spoofed attack

  DNS servers cannot tell by examining a particular packet wether the source address in that

packet is real or not. Most *DNS* queries are done by UDP. UDP does not have source address verification.

- Small *DNS* queries can generate large responses

  Especially when used with *DNSSEC*, the responses can be 10-20 (or more) times larger than the question.

10.4 A solution

If one packet with a forged source address arrives at a DNS server, there is no way for the server to tell it is forged. If hundreds of packets per second arrive with very similar source addresses asking for similar or identical information, there is a very high probability that those packets, as a group, form part of an attack. The Response Rate Limiting (RRL) algorithm has two parts. It detects patterns in incoming queries, and when it finds a pattern that suggests abuse, it can reduce the rate at which replies are sent.

- Clients are grouped by their masked IPs, using **ipv4-prefix-length** and **ipv6-prefix-length**.
- Clients are kept in a table with a size varying from **min-table-size** to **max-table-size**.
- The **responses-per-second** is the maximum number of "no-error" answers that will be given to a client in the duration of a second.
- The **errors-per-second** is the maximum number of error answers that will be given to a client in the duration of a second.
- The **window** is the period for which the rates are measured. If the client goes beyond any of its allowed rates, then the majority of further answers will be dropped until this period of time has elapsed. Every **slip** dropped answers, a truncated answer may randomly be given, llowing the client to ask the query again using TCP.

11 Multi Master

11.1 Introduction

A multi-master *DNS* server configuration is a setup where more than one primary name server exists for the same zone and a secondary name server is configured to communicate with multiple primary name servers.

The benefit of having a multi-master configuration is that if one of the primary name servers is down or is in a maintainance mode the secondary name server can still request updates.

The secondary name server will listen to the notifications from all the primary name servers, but will always request the updates from the same preferred primary name server. When the preferred name server is unable to provide correct services, the next primary name server in the list of primary name servers (**masters**) will be used. From then on, this primary name server has the highest priority in the list and becomes the new preferred primary name server.

11.1.1 Design

Wheterh a slave is configured with a single master or with multiple masters, the design remains similar. The differences for the multi-master design will be highlighted in this section of the manual.

Single master

When a slave zone has a single master configured, *YADIFA* will check the *SOA* serial on disk and request an IXFR from this serial to the (only) primary name server. If no files exist on disk, *YADIFA* will initiate an AXFR. When the transfer is successful, the zone is loaded. When notifications are received from the master, it will check the serial in the notification and when the serial is absent or higher, *YADIFA* will initiate an *IXFR* with the current serial to the master.

When a transfer error occurs, *YADIFA* will try to contact the primary name server again after a delay. The backing-off mechanism is explained in a different section.

configuration example

```
<zone>
    domain      somedomain.eu
    file        slaves/somedomain.eu.
    type        slave
    masters     192.0.2.1
</zone>
```

Multiple masters

When a slave zone has multiple masters configured, *YADIFA* will use the first configured master as the preferred primary name server. In normal operations, it will behave identical to when only a single primary name server is defined with one minor difference. Notifications received from different (not the preferred) primary masters, will be trigger the normal transfer procedure to the preferred master. If the preferred primary name server itself is lagged with updates, *YADIFA* will not try to find the most current server (highest serial), but keep itself in sync with the prefered master. This is a deliberate design decision and will be explained later in this document.

The differences become apparent when a zone transfer fails. When the number of transfer failures exceed the **multimaster-retries** option, the next primary name server will be selected as the new preferred master. The previous preferred master is added to the end of the list. The backing-off mechanism is explained in a different section.

note

A *Notification of Zone Changes*[49] (*DNS NOTIFY*) from a different primary will trigger the mechanism to update the zone, but *YADIFA* will keep itself in sync with the preferred primary only.

note

With the exception of a reload of the configuration, transfer failures will be considered as the only reason to change the preferred master.

The reason, be it a networking error, *Server Failure (rcode 2)*[44] (*SERVFAIL*), *Server Not Authoritative for zone (rcode 9)*[14] (*NOTAUTH*), *TSIG*, too slow, or anything else causing a transfer failure, is irelevant for the switching decision. When the transfer is not successful, it is considered a failure.

configuration example

```
<zone>
    domain               somedomain.eu
    file                 slaves/somedomain.eu.
    type                 slave
    masters              192.0.2.1,192.0.2.2,192.0.2.3
    multimaster-retries 2
</zone>
```

In this example:

- The list of primaries is "192.0.2.1,192.0.2.2,192.0.2.3" and the first preferred primary is 192.0.2.1 and will be used to initiate a transfer of the zone.
- When a *DNS NOTIFY* is received from any primary (e.g. 192.0.2.2) the *SOA* of the preferred master 192.0.2.1 is checked. If the serial is bigger a transfer will be initiated from 192.0.2.1.
- If the transfer from 192.0.2.1 fails 3 times (initial + 2 retries), the next primary in the list (192.0.2.2) will become the new preferred primary and the new list will be "192.0.2.2,192.0.2.3,192.0.2.1".

note

When true-multimaster is set to false (default), the secondary name server will not perform a (partial) zone transfer when switching to the new preferred master with a lower or identical serial.

True multimaster

There are several scenarios in which an organisation runs several independent primary name servers. When there are independent masters, we cannot be sure that the zone content on all the primaries is identical. Differences in update sizes or in jitter may cause differences in the zone content. The flag **true-multimaster** should be used in this case.

The behavior of *YADIFA* is similar to a regular multimaster setup, with the difference that, when a new preferred primary is taken, the system will request a full zone transfer rather than an incremental. Updates from the same preferred primary will result in an *IXFR*. Switching to a different preferred primary should be avoided as it would otherwise result in a lot of unnecessary strain on the primaries, the secondaries and the network. Therefore, when a notify is received from a primary name server which is not the preferred primary, the serial of the preferred primary is checked. And an incremental transfer is initiated from the preferred master when necessary.

note

When true-multimaster is set to true, the secondary name server will always perform a full zone transfer when switching to the new preferred master regardless of the serial number.

Backing-off mechanism

The back-off time before a new transfer is attempted, can be configured in the <*main*> section with the option **xfr-retry-delay**. A jitter can also be applied with the option **xfr-retry-jitter**. To increase the back-off time between failed transfers, two other parameters can be used: **xfr-retry-failure-delay-multiplier** and **xfr-retry-failure-delay-max**.

The formula for the backing-off mechanism is the following:

XFR-RETRY-DELAY + XFR-RETRY-JITTER + MIN(FAILED-TRANSFERS * XFR-RETRY-FAILURE-DELAY-MULTIPLIER ; XFR-RETRY-FAILURE-DELAY-MAX)

configuration

```
<main>
    xfr-retry-jitter                    0       # Not possible, minimum 60
                                                # but makes the math clearer
    xfr-retry-delay                     200
    xfr-retry-failure-delay-multiplier  50
    xfr-retry-failure-delay-max         200
</main>

<zone>
    domain              somedomain.eu
    file                slaves/somedomain.eu.
    type                slave
    masters             192.0.2.1,192.0.2.2,192.0.2.3
    multimaster-retries 6
</zone>
```

In this example:

The **xfr-retry-jitter** is ignored to make the example easier to explain.

Consider the following scenario:

- The preferred primary is 192.0.2.1 is unavailable, as is 192.0.2.2.
- An update to the zone is done.

- A *DNS NOTIFY* is received from 192.0.2.3.

YADIFA will do the following:

1. check the *SOA* over UDP with the preferred master 192.0.2.1 which fails.
2. initiate an *IXFR* with the current serial over TCP which also fails.
3. *YADIFA* will wait 250 seconds (200 + 1 * 50) (first failure) which also fails.
4. *YADIFA* will wait 300 seconds (200 + 2 * 50) (second failure) which also fails.
5. *YADIFA* will wait 350 seconds (200 + 3 * 50) (third failure) which also fails.
6. *YADIFA* will wait 400 seconds (200 + 4 * 50) (fourth failure) which also fails.
7. *YADIFA* will wait 400 seconds (200 + 200) (fifth failure) which also fails.
8. *YADIFA* will wait 400 seconds (200 + 200) (sixth failure) which also fails.
9. *YADIFA* will wait 400 seconds and switch the preferred primary to 192.0.2.2 and transfer fails.
10. *YADIFA* will wait 250 seconds (200 + 1 * 50) (first failure) which also fails.
11. *YADIFA* will wait 300 seconds (200 + 2 * 50) (second failure) which also fails.
12. *YADIFA* will wait 350 seconds (200 + 3 * 50) (third failure) which also fails.
13. *YADIFA* will wait 400 seconds (200 + 4 * 50) (fourth failure) which also fails.
14. *YADIFA* will wait 400 seconds (200 + 200) (fifth failure) which also fails.
15. *YADIFA* will wait 400 seconds (200 + 200) (sixth failure) which also fails.
16. *YADIFA* will wait 400 seconds and switch the preferred primary to 192.0.2.3 and transfer succeeds.

Design reasoning

The design of *YADIFA* takes the following into consideration, in order of importance:

1. The integrity of the zone content
2. The availability of the zone.
3. The zone content is up-to-date.

In a secondary name server, there are 9 possible areas in which a zone file can be:

1. True Master ON in the zone section of the secondary name server

   - Zone data, where the primary name server uses Dynamic Updates to update content and the zone file is *DNSSEC*
   - Zone data, where the primary name server uses Dynamic Updates to update content
   - Zone data, where the primary name server does not use Dynamic Updates, the content is updated through the reloading of the zone data and an augmentation of the serial of the *SOA*
   - Zone data, where the primary name server does not use Dynamic Updates and the zone data is *DNSSEC*, the content is updated through the reloading of the zone data and an augmentation of the serial of the *SOA*.

The **true-master** option in *YADIFA* is used for installations where the zone content of the primary name servers is not identical. The reasons as to why the zone content is not identical is beyond the scope of this document.

Defining multiple primary masters for a zone file indicates that, if the secondary name server is unable to transfer the zone from the preferred primary name server, the secondary name server will communicate with the next primary name server in its list of masters for reception of its zone content.

As the zone content is not guaranteed to be identical, the only option is to perform a full transfer. With that said, as changing between primary masters is very costly resource wise, *YADIFA* allows, tuneable with several parameters, for the preferred primary to recover from any temporary issues that might otherwise lead to a switch. Altough networks have become very reliable, a *DNS NOTIFY* is sent through UDP which does not guarantee delivery. Therefore, when a *DNS NOTIFY* from a different primary is received, *YADIFA* will still check the *SOA* serial of the preferred primary in case the notify was lost.

In all the cases, (dynamic, static, *DNSSEC* or not *DNSSEC*), delaying a switch to a different primary master will reduce the amount of wasted resources while maintaining the highest operational performance. The connection retries to the primary name server can be configured accordingly. If, after 'X' retries no connection can be established with the primary name server, the second primary name server will take its place in the list, resulting in an *AXFR*.

note

```
In true multi-master setups, the same or a higher serial does not mean that
the zone content is more up-to-date.
```

2. True Master OFF

In this case, YADIFA considers all the primary name servers with the same serial as having identical zone data.

- Zone data, where the primary name server uses Dynamic Updates to update content and the zone file is *DNSSEC*
- Zone data, where the primary name server uses Dynamic Updates to update content
- Zone data, where the primary name server does not use Dynamic Updates, the content is updated through the reloading of the zone data and an augmentation of the serial of the *SOA*
- Zone data, where the primary name server does not use Dynamic Updates and the zone data is *DNSSEC*, the content is updated through the reloading of the zone data and an augmentation of the serial of the *SOA*
- Zone data, where there is a single primary name server and intermediary masters.

When *YADIFA* receives a *DNS NOTIFY*, it always communicates with the same primary name server for reception of the changes (*AXFR* or *IXFR*). If *YADIFA* receives a *DNS NOTIFY* that contains a *SOA* resource record with a lesser or equal serial than its own, it ignores the message.

However, for primary name servers using a dynamic zone file with *DNSSEC*, one REALLY cannot be sure, no matter the configuration, that the same *SOA* serial has the same zone data. This is due to jitter in signing the zone, resigning of the zone and dynamic updates which are never completely on the same time on all primary name servers. This results in the content not being 100 percent identical on all the primary name servers. In this case, **true-master** ON is the best and only choice. Please Note: This relates to real primary name servers and not intermediary masters.

In the other cases, assuming for the *DNSSEC* enabled zone that all the signatures are pre-calculated and that primary name server(s) are not responsible for maintaining the signatures, which would otherwise result in a scenario where true multi-master would be preferred, we are absolutely sure that the content is identical. The zone content could be updated quickler by switching to the first primary name server for which a *DNS NOTIFY* is received.

If switching to a different primary name server could be performed with an incremental transfer, the cost of switching would be negligable and would result in the most up-to-date information for the slave. Unfortunately, we cannot be sure that switching to a different primary will result in a small incremental transfer.

Some setups (e.g. without bind's **ixfr-from-differences** yes;) could result in an *AXFR* through an update while others have huge incremental updates. The primary name servers in the configuration may have other paths with different bandwidth restrictions and costs associated with them. Therefor the benefits of quickly switching to a different primary is uncertain therefore, the choice is given to the administrator to specify the most desirable primary. *YADIFA* will respect this choice by only switching when absolutely necessary.

For a host master with thousands of zones to administer, the load between different masters can be distributed by simply rotating the primary name servers in the configuration.

3. Round robin scheme vs original preference list

To avoid flapping services, we have opted to implemented a round-robin scheme. When the first primary name server is known to be bad (configurable), the next primary name server in the list will become the new preferred primary. When the host master has addressed the issue and wants to switch back to the "first" primary name server, this can be done by issuing a config reload.

11.2 What is needed?

11.2.1 Zone

configuration example of *<zone>* with several masters

```
<zone>
    domain              somedomain.eu
    file                somedomain.eu.
    type                slave

    masters             192.0.2.1,192.0.2.2,192.0.2.3
    true-multimaster    no  # 'no' is default, this line can be left out
</zone>
```

In this example the secondary name server listens to the notifications from the 3 primary name servers (masters). The secondary name server will always ask for *DNS UPDATE*S from the first in the list. In this example, 192.0.2.1.

If the first primary name server no longer answers, the secondary name server will ask for updates from the second primary name server in the list, 192.0.2.2. From then on the secondary name server continues to ask that primary name server for updates until it no longer answers. Once that happens the secondary name server asks the next one in the list. After the last primary name server stops answering, the secondary name server starts from the first in the list, 192.0.2.1, again.

If the **true-multimaster** is set to "no", the secondary name server expects that all primary name servers are in sync and that their zone information is the same.

12 Configuration Reference

12.1 Layout

The configuration file has some rules:

- The configuration is read from a simple text file.
- A comment starts after the '#' character.
- Empty lines have no effect.
- A string can be double quoted, but is not mandatory.

The configuration file is made up of sections. A section starts with a with a <*name*> line and ends with a </*name*> line.

Currently the following sections are implemented:

- <*main*>
- <*zone*>
- <*key*>
- <*acl*>
- <*channels*>
- <*loggers*>
- <*nsid*>
- <*rrl*>
- <*dnssec-policy*>
- <*key-suite*>

- *<key-roll>*
- *<key-template>*
- *<denial>*

Unimplemented section names are ignored.

The section order is only of importance for sections of the same type where the principle first-found-first-processed applies. In other words, the last settings will overwrite ealier declarations of the same parameter. One exception is the *<zone>* section, where a declaration for the same domain will result in the error **DATABASE_ZONE_CONFIG_DUP**.

configuration example

```
<zone>
    domain  somedomain.eu
    file    masters/somedomain.eu.zone
    type    master
</zone>

<zone>
    domain  somedomain.eu
    file    masters/somedomain2.eu.txt
    type    master
</zone>
```

In this example for the zone *somedomain.eu*, the *file* will be "masters/somedomain.eu.zone".

The processing order of each section type is determined by the server implementation.
Each section contains settings. A setting is defined on one line but can be spread over multiple lines using parenthesis.

configuration example

```
# comment
# comment
<first>
# commment
    setting0-name value ...
    setting1-name value ...
</first>

<second>
    setting2-name (
        value
        ...
    )
# comment
</second>
```

12.2 Types

Each setting can be one of the following types.

TYPE	DESCRIPTION
ACL	A list of ACL descriptors. User-defined ACLs are found in the <*acl*> section. The 'any' and 'none' descriptors are always defined. Elements of the list are separated by a ',' or a ';'.
DNSSECTYPE	A *DNSSEC* type name. It can be a DNSSEC-enabled value ("nsec", "nsec3" or "nsec3-optout") or a DNSSEC-disabled value ("none", "no", "off" or "0").
ENUM	A word from a specified set.
FLAG	A boolean value. It can be true ("1", "enable", "enabled", "on", "true", "yes") or false ("0", "disable", "disabled", "off", "false", "no").
FQDN	An **Fully Qualified Domain Name** (FQDN) text string. i.e.: www.eurid.eu.
GID	Group ID. (Can be a number or a name)
HOST(S)	A (list of) host(s). A host is defined by an IP (v4 or v6) and can be followed by the word 'port' and a port number. Elements of the list are separated by a ',' or a ';'.
INTEGER / INT	A base-ten integer.
PATH / FILE	A file or directory path. i.e.: '/var/zones'.
STRING / STR	A text string. Double quotes can be used but are not mandatory. Without quotes the string will be taken from the first non-blank charater to the last non-blank character.
UID	User ID. (Can be a number or a name)

TYPES

12.3 Sections

12.3.1 <*main*> section

This section defines the global or default settings of the server.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
allow-control	ACL	none	Default server-control access control list. Only the sources matching the ACL are accepted.
allow-notify	ACL	any	Default notify access control list. Only the servers matching the ACL will be handled.
allow-query	ACL	any	Default query access control list. Only the clients matching the ACL will be replied to.
allow-transfer	ACL	none	Default transfer access control list. Only the clients matching the ACL will be allowed to transfer a zone (*AXFR*/*IXFR*).
allow-update	ACL	none	Default update access control list. Only the clients matching the ACL will be allowed to update a zone.
allow-update-forwarding	ACL	none	Default update-forwarding access control list. Only the sources matching the ACL are accepted.
answer-formerr-packets	FLAG	true	If this flag is disabled; the server will not reply to badly formatted packets.
axfr-compress-packets; axfr-compresspackets	FLAG	true	Enables the *DNS* packet compression of each *AXFR* packet.
axfr-max-packet-size; axfr-maxpacketsize	INT	4096 bytes	The maximum size of an *AXFR* packet. (MIN: 512; MAX: 65535)
axfr-max-record-by-packet; axfr-maxrecordbypacket	INT	0	The maximum number of records in each *AXFR* packet. Older name servers can only handle 1. Set to 0 to disable the limit. (MIN: 0; MAX: 65535)
axfr-retry-delay;xfr-retry-delay	INT	600 sec	Number of seconds between each retry for the first transfer from the master name server. (MIN: 60; MAX: 86400)
axfr-retry-jitter;xfr-retry-jitter	INT	180 sec	Jitter applied to axfr-retry-delay. (MIN: 60; MAX: axfr-retry-delay)
axfr-retry-failure-delay-multiplier;xfr-retry-failure-delay-multiplier	INT	5	Linear back-off multiplier. The multiplier times the number of failures is added to the xfr-retry-delay. (MIN: 0; MAX: 86400)
axfr-retry-failure-delay-max;xfr-retry-failure-delay-max	INT	3600	Maximum delay added for the back-off. (MIN: 0; MAX: 604800)
chroot	FLAG	off	Enabling this flag will make the server jail itself in the chroot-path directory.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
chroot-path; chrootpath	PATH	/	The directory used for the jail.
cpu-count-override	INT	0	Overrides the detected number of logical cpus. Set to 0 for automatic. (MIN: 0; MAX: 256)
config-file; configfile	FILE	yadifad.conf[a]	The configuration file.
daemon; daemonize	FLAG	false	Enabling this flag will make the server detach from the console and work in background.
data-path; datapath	PATH	zones[b]	The base path were lies the data (zone file path; journaling data; temporary files; etc.)
dnssec-thread-count	INT	0	The maximum number of threads used for *DNSSEC* parallel tasks (mostly signatures). Set to 0 for automatic. (MIN: 0; MAX: 128)
edns0-max-size	INT	4096	*EDNS0* packets size. (MIN: 512; MAX: 65535)
gid; group	GID	0 (or root)	The group ID that the server will use.
hostname-chaos; hostname	STR	the host name	The string returned by a hostname.chaos TXT CH query.
keys-path; keyspath	PATH	zones/keys[b]	The base path of the *DNSSEC* keys.
listen	HOSTS	0.0.0.0	The list of interfaces to listen to.
log-from-start	FLAG	off	Enabling this flag will make the server start logging immediately to stdout; even before the loggers are initialized.
log-path; logpath	PATH	log[b]	The base path where the log files are written.
log-unprocessable	FLAG	off	Enabling this flag will make the server log unprocessable queries.
max-tcp-queries; max-tcp-connections	INT	16	The maximum number of parallel TCP queries; allowed. (MIN: 1; MAX: 255)
pid-file; pidfile	STR	run/yadifad.pid[b]	The pid file name.
queries-log-type	INT	1	Query log format. (0: none; 1: *YADIFA* format; 2: BIND format; 3: YADIFA and BIND format at once)
serverid-chaos; serverid	STR	-	The string returned by a id.server. TXT CH query. If not set; REFUSED is answered.

MAIN SECTION

[a] SYSCONFDIR is set at compile time; typically PREFIX/etc or /etc/yadifa
[b] LOCALSTATEDIR is set at compile time; typically PREFIX/var or /var

PARAMETER	TYPE	DEFAULT	DESCRIPTION
server-port; port	INT	53	The default *DNS* port. (MIN: 1; MAX: 65535)
sig-signing-type	INT	65534	The resource record type that will be created while the signing is in progress. (MIN: 0; MAX: 65535)
sig-validity-interval	INT	30 (days)	The number of days for which an automatic signature is valid. (MIN: 7 days; MAX: 30 days)
sig-validity-jitter; sig-jitter	INT	3600 sec	The signature expiration validity jitter in seconds (1 hour). (MIN: 0 sec; MAX: 86400 sec)
sig-validity-regeneration	INT	auto hours	Signatures expiring in less than the indicated amount of hours will be recomputed. The default will be chosen by *YADIFA* . (MIN: 24 hours; MAX: 168 hours)
statistics	FLAG	true	The server will log a report line about some internal statistics.
statistics-max-period	INT	60 sec	The period in seconds between two statistics log lines. (MIN: 1 sec; MAX: 31 * 86400 seconds (31 days))
tcp-query-min-rate	INT	512 bytes/sec	The minimum transfer rate required in a TCP connection (read and write). Slower connections are closed. The units are bytes per second. (MIN: 0; MAX: 4294967295)
thread-count-by-address	INT	-1	Number of independent threads used to process each listening address. Set to -1 for automatic. Set to 0 for single threaded. (MIN: -1; MAX: number of CPU's)
uid; user	UID	0 (or root)	The user ID that the server will use.
version-chaos; version	STR	yadifa version#	The string returned by a version TXT CH query.
xfr-connect-timeout	INT	5 sec	Timeout for establishing a connection for *AXFR* and *IXFR* transfers. (MIN: 0 sec; MAX: 4294967295 sec)
xfr-path; xfrpath	PATH	zones/xfr[b]	The base path used for *AXFR* and journal storage.
zone-load-thread-count	INT	1	Number of independent threads used to process loading of the zones. (MIN: 0; MAX: 4294967295)
zone-download-thread-count	INT	4	Number of independent threads used to download the zones. (MIN: 0; MAX: 4294967295)

MAIN SECTION

configuration example

```
<main>
    chroot                      on
    daemonize                   true
    chroot-path                 /srv/yadifa/var
    keys-path                   /zones/keys
    data-path                   /zones
    log-path                    /log
    pid-path                    /run
    pid-file                    yadifad.pid

    cpu-count-override          6
    dnssec-thread-count         10
    max-tcp-queries             100
    tcp-query-min-rate          6000

    additional-from-auth        yes
    authority-from-auth         yes
    answer-formerr-packets      no

    listen                      192.0.2.53, 192.0.2.153 port 8053

    hostname                    my-shown-hostname
    serverid                    ns-loc-01

    user                        yadifad
    group                       yadifad

    statistics                  yes
    statistics-max-period       60

    # could have been written as: 'version not disclosed' without the '
    version                     "not disclosed"

    # note: Any is default anyway
    allow-query                 any
    allow-update                operations-network ; public-network
    allow-transfer              slaves ; operations-network ; public-network

    sig-signing-type            65542
    sig-validity-interval       360
    sig-validity-regeneration   48
    sig-validity-jitter         1800

    axfr-max-record-by-packet   0
    axfr-max-packet-size        32768
    axfr-compress-packets       true
</main>
```

12.3.2 <*zone*> sections

Each zone is defined by one section only.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
allow-control	ACL	as main	Control commands control list. Only the matching sources are allowed.
allow-notify	ACL	as main	Notify access control list. Only the servers matching the ACL will be handled.
allow-query	ACL	as main	Query access control list. Only the clients matching the ACL will be replied to.
allow-transfer	ACL	as main	Tansfer access control list. Only the clients matching the ACL will be allowed to transfer a zone (*AXFR/IXFR*
allow-update	ACL	as main	Update access control list. Only the clients matching the ACL will be allowed to update a zone.
allow-update-forwarding	ACL	as main	Update forwarding control list. Only the matching sources are allowed.
dnssec-mode; dnssec	DNSSEC-TYPE	off	Type of *DNSSEC* used for the zone. As master name sever; *YADIFA* will try to maintain that state.
dnssec-policy	STR	-	Sets the dnssec-policy id to be used.
domain	FQDN	-	Mandatory. Sets the domain of the zone (i.e.: eurid.eu).
drop-before-load	FLAG	off	Enabling this flag will make the server drop the zone before loading the updated zone from disk. Use this on systems constrained for RAM.
file-name; file	FILE	-	Sets the zone file name. Only mandatory for a master zone. Relative paths to <*main*> **data-path**
journal-size-kb; journal-size	INT	0	Puts a soft limit on the size of the journal; expressed in KB. (MIN: 0; MAX: 8388608 (8GB))
keys-path; keyspath	PATH	as main	The base path of the *DNSSEC* keys.
maintain-dnssec	FLAG	true	Enabling this flag will cause the server to try and maintain *RRSIG* records
masters; master	HOSTS	-	Mandatory for a slave. Sets the master server(s). Multiple masters are supported.

ZONE SECTION

PARAMETER	TYPE	DEFAULT	DESCRIPTION
multimaster-retries	INT	0	The number of times the master is unreachable before switching to a different master. (MIN: 0; MAX: 256)
no-master-updates	FLAG	false	Enabling this flag will prevent the server from probing or downloading changes from the master.
notifies; also-notify; notify	HOSTS	-	The list of servers to notify in the event of a change. Currently only used by masters when a dynamic update occurs.
notify-auto	FLAG	true	Enabling this flag will cause *DNS NOTIFY* messages to be sent to all name servers in the APEX. Disabling this flags causes the content of APEX to be ignored (*NS* Records).
notify-retry-count; retry-count	INT	5	Number of times *YADIFA* tries to send a *DNS NOTIFY*. (MIN: 0; MAX: 10)
notify-retry-period; retry-period	INT	1	Time period in minutes between two *DNS NOTIFY* attempts. (MIN: 1; MAX: 600)
notify-retry-period-increase; retry-period-increase	INT	0	Increase of the time period in minutes between two *DNS NOTIFY* attempts. (MIN: 0; MAX: 600)
sig-validity-interval; signature-validity-interval	INT	as main	The number of days for which an automatic signature is valid. (MIN: 7 days; MAX: 30 days)
sig-validity-regeneration; signature-regeneration	INT	as main	The signatures expiring in less than the indicated amount of hours will be recomputed. (MIN: 24 hours; MAX: 168 hours)
sig-validity-jitter; signature-sig-jitter; signature-jitter; sig-jitter	INT	as main	The signature expiration validity jitter in seconds. (MIN: 0 sec; MAX: 86400 sec)
true-multimaster	FLAG	off	Enabling this flag will make the server use *AXFR* when switching to a new master.
type	ENUM	-	Mandatory. Sets the type of zone : either **master** or **slave**.

ZONE SECTION

sig-* and allow-* settings defined here have precedence over those in the <*main*> section.

configuration example

```
<zone>
    domain                      somedomain.eu.
    type                        master
    file-name                   masters/somedomain.eu-signed.txt

    # The rest is not mandatory ...

    also-notify                 192.0.2.194, 192.0.2.164

    # Doing this is pointless since it's both the global setting AND
    # the default one

    allow-query                 any
    allow-update                my-network; 127.0.0.1
    allow-transfer              my-slaves

    # Same as global setting
    sig-signing-type            65542
    sig-validity-interval       720             # 30 days is enough
    sig-validity-regeneration   12
    sig-validity-jitter         7200

    journal-size-kb             64               # 64 KB
</zone>

<zone>
    domain                      another-zone.eu
    type                        slave
    master                      192.0.2.53
</zone>
```

12.3.3 <*key*> sections

Each TSIG key must be defined by one section.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
algorithm	ENUM	-	Mandatory. Sets the algorithm of the key. Supported values are : 'hmac-md5', 'hmac-sha1', 'hmac-sha224', 'hmac-sha256', 'hmac-sha384', 'hmac-sha512' (the algorithm names are case insensitive)
name	FQDN	-	Mandatory. Sets the name of the key.
secret	TEXT	-	Mandatory. Sets the value of the key. BASE64 encoded.

KEY SECTION

configuration example

```
<key>
    name            yadifa
    algorithm       hmac-md5
    secret          WouldNtYouWantToKnowIt==
</key>

<key>
    name            eu-slave1
    algorithm       hmac-md5
    secret          WouldNtYouWantToKnowIt==
</key>

<key>
    name            eu-slave2
    algorithm       hmac-md5
    secret          WouldNtYouWantToKnowIt==
</key>
```

12.3.4 <*acl*> section

Each entry of the acl section defines a rule of access. Each rule is a name (a single user-defined word) followed by a rule in the form of a list of statements. The separator can be ',' or ';'. The 'any' and 'none' names are reserved. A statement tells if a source is accepted or rejected. Reject statements are prefixed with '!'. Statements are evaluated in the following order: first from more specific to less specific, then from reject to accept. If a statement matches, the evaluation will stop

and accordingly accept or reject the source. If no statement matches, then the source is rejected.

A statement can be either:

- An IPv4 or an IPv6 address followed (or not) by a mask.
  [!]ipv4|ipv6[/mask]

  For example:

  configuration sample

  ```
  internal-network            192.0.2.128/26;2001:DB8::/32
  ```

- The word 'key' followed by the name of a TSIG key.
  key key-name

  For example:

  configuration sample

  ```
  slaves                      key public-slave;key hidden-slave
  ```

- An ACL statement name from the <*acl*> section. Note that negation and recursion are forbidden and duly rejected.
  acl-name

  For example:

  configuration sample

  ```
  who-can-ask-for-an-ixfr     master;slaves;127.0.0.1
  ```

configuration example

```
<acl>
    # user-defined-name    rule-statements

    # rule to accept this TSIG key

    slave1                  key eu-slave1

    # rule to accept that TSIG key

    slave2                  key eu-slave2

    # rule to accept what the slave1 and slave2 rules are accepting

    slaves                  slave1;slave2

    # rule to accept this IP
    master                  192.0.2.2

    # rule to accept both this IPv4 network and that IPv6 network
    operations              192.0.2.128/28;2001:DB8::/32

    # Now about the order of each ACL statement : the following rule

    order-example-1         192.0.2.128/26 ; 192.0.2.5 ;
                            ! 192.0.2.133 ; ! 192.0.2.0/26

    # will be understood the same way as this one

    order-example-2         192.0.2.5 ; !192.0.2.133 ;
                            192.0.2.128/26 ; !192.0.2.0/26

    # Because in effect, both will be seen internally as:

    order-example-3         !192.0.2.133 ; 192.0.2.5 ;
                            !192.0.2.0/26 ; 192.0.2.128/26

</acl>
```

12.3.5 <*channels*> section

Channels are loggers output stream definitions. Three types are supported:

- file

- STDOUT, STDERR
- syslog.

Each channel is a name (a single user-defined word) followed by:

- the 'syslog' keyword, defining a channel to the syslog daemon. The keyword can be followed by case-insensitive facilities and options arguments. These arguments will be given to syslog. Note that only one facility should be given.

  Supported facilities:

PARAMETER	DESCRIPTION
auth	Security/authorisation messages (DEPRECATED: use authpriv)
authpriv	Security/authorisation messages (private)
cron	Clock daemon (cron and at)
daemon	System daemons without separate facility value
ftp	Ftp daemon
local0	Reserved for local use
local1	Reserved for local use
local2	Reserved for local use
local3	Reserved for local use
local4	Reserved for local use
local5	Reserved for local use
local6	Reserved for local use
local7	Reserved for local use
lpr	Line printer subsystem
mail	Mail subsystem
news	USENET news subsystem
syslog	Messages generated internally by syslogd(8)
user	Generic user-level messages
uucp	UUCP subsystem

CHANNELS SECTION

  Supported options:

PARAMETER	DESCRIPTION
cons	Write directly to system console if there is an error while sending to system logger.
ndelay	Open the connection immediately (normally, the connection is opened when the first message is logged).
nowait	Don't wait for child processes that may have been created while logging the message (On systems where it is relevant).
odelay	Opening of the connection is delayed until syslog() is called (This is the default, and need not be specified).
perror	(Not in POSIX.1-2001.) Print to stderr as well.
pid	Include PID with each message.

CHANNELS (syslog) SECTION

note

```
For more information:  man syslog
```

For example:

configuration sample

```
syslog  syslog  CRON,PID
```

- The 'STDOUT' case-sensitive keyword, defining a channel writing on the standard output.

  For example:

  configuration sample

  ```
  default-output STDOUT
  ```

- The 'STDERR' case-sensitive keyword, defining a channel writing on the standard error.

  For example:

  configuration sample

  ```
  default-error STDERR
  ```

- A relative file path, defining a channel writing on a file (append at the end). The file is followed by the file rights as an octal number.

  For example:

  configuration sample

  ```
  yadifa yadifa.log 0644
  ```

configuration example

```
<channels>
    # user-defined-name     parameters

    # channel 'statistics': a file called stats.log
    #                       with 0644 access rights
    #
    statistics              stats.log 0644

    # channel 'syslog' :    a syslog daemon output using
    # the local6 facility and logging the pid of the process
    #
    syslog                  syslog local6,pid

    # channel 'yadifa': a file called yadifa.log with 0644 access rights
    #
    yadifa                  yadifa.log 0644

    # channel 'debug-out' : directly printing to stdout
    #
    debug-out               STDOUT

    # channel 'debug-err' : directly printint to stderr
    #
    debug-err               STDERR
</channels>
```

12.3.6 <*loggers*> section

Yadifa has a set of log sources, each of which can have their output filtered (or ignored) and sent to a number of channels.

A logger line is defined as the source name followed by the list of levels and then the list of channels. The lists are ',' separated.

The current set of sources is:

SOURCES	DESCRIPTION
database	Database output (incremental changes, integrity checks, etc.)
dnssec	*DNSSEC* output (*NSEC*, *NSEC3*, signatures events)
server	Server actions output (network setup, database setup, queries, etc.)
stats	Internal statistics periodic output
system	Low-level output (thread management, task scheduling, timed events)
zone	Internal zone loading output
queries	Queries output

LOGGERS SECTION

The current set of levels is:

LEVELS	DESCRIPTION
emerg	System is unusable
alert	Action must be taken immediately
crit	Critical conditions
err	Error conditions
warning	Warning conditions
notice	Normal, but significant, condition
info	Informational message
debug	Debug-level 0 message
debug1	Debug-level 1 message
debug2	Debug-level 2 message
debug3	Debug-level 3 message
debug4	Debug-level 4 message
debug5	Debug-level 5 message
debug6	Debug-level 6 message
debug7	Debug-level 7 message
prod	All non-debug levels
all	All levels
*	All levels

LEVELS

note

`Messages at the ‘crit’, ‘alert’ and ‘emerg’ levels do trigger an automatic shutdown of the server.`

If the logger section is omitted completely, everything is logged to the STDOUT channel. Negations are not allowed.

configuration

```
<loggers>
    # info, notice and warning level messages from the database logging
    # will be output
    database        info,notice,warning     yadifa
    database        err,crit,alert,emerg    yadifa,syslog
    server          *                       yadifa
    stats           *                       statistics
    system          *                       debug-err
    queries         *                       queries
    zone            *                       yadifa
</loggers>
```

The defined loggers are:

system contains low level messages about the system such as memory allocation, threading, IOs, timers and cryptography, ...

database contains messages about most lower-level operations in the DNS database. ie: journal, updates, zone loading and sanitization, DNS message query resolution, ...)

dnssec contains messages about lower-level dnssec operations in the DNS database. ie: status, maintenance, verification, ...

server contains messages about operations in the DNS server. ie: startup, shutdown, configuration, transfers, various services status (database management, network management, DNS notification management, dynamic update management, resource rate limiting, ...)

zone contains messages about the loading of a zone from a source (file parsing, transferred binary zone reading, ...)

stats contains the statistics of the server. (See chapter 15)

queries contains the queries on the server. Queries can be logged with the bind and/or with the *YADIFA* format.

bind format:

client sender-ip#port: query: fqdn class type +SETDC (listen-ip)

***YADIFA* format:**

query [id] {+SETDC} fqdn class type (sender-ip#port)

where:

id is the query message id

+ means the message has the Recursion Desired flag set

S means the message is signed with a *TSIG*

E means the message is EDNS

T means the message was sent using TCP instead of UDP

D means the message has the DNSSEC OK flag set

C means the message has the Checking Disabled flag set

fqdn is the queried FQDN

class is the queried class

type is the queried type

sender-ip is the IP of the client that sent the query

port is the port of the client that sent the query

listen-ip is the listen network interface that received the message

Note that on *YADIFA* any unset flag is replaced by a '-', on bind only the '+' follows that rule.

System operators will mostly be interested in the info and above messages of queries and stats, as well as the error and above messages of the other loggers.

12.3.7 <*nsid*> section

note

If you want to have DNS Name Server Identifier Option (NSID) support in *YADIFA* you need to enable this function before compiling the sources.

shell

```
$ ./configure --enable-nsid
```

After the 'configure', you can do the normal 'make' and 'make install'.

shell

```
$ make
$ make install
```

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ascii hex	STR	"" ""	The string can be 512 characters long.

NSID SECTION

configuration example ascii

```
<nsid>
    ascii belgium-brussels-01
</nsid>
```

configuration example hex

```
<nsid>
    hex   00320201
</nsid>
```

12.3.8 <*rrl*> section

YADIFA has support for RRL enabled by default.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
responses-per-second	INT	5	Allowed response rate.
errors-per-second	INT	5	Allowed error rate.
slip	INT	2	Random slip parameter.
log-only	FLAG	false	If set to true, logs what it should do without doing it.
ipv4-prefix-length	INT	24	Mask applied to group the IPv4 clients.
ipv6-prefix-length	INT	56	Mask applied to group the IPv6 clients.
exempt-clients,exempted	ACL	none	Clients maching this rule are not subject to the RRL.
enabled	FLAG	false	Enables the RRL
min-table-size	INT	1024	RRL buffer minimum size
max-table-size	INT	16384	RRL buffer maximum size
window	INT	15	RRL sliding window size in seconds

RRL SECTION

configuration example

```
<rrl>
    responses-per-second        5
    errors-per-second           5
    slip                        10
    log-only                    off
    ipv4-prefix-length          24
    ipv6-prefix-length          56
    exempt-clients              none
    enabled                     yes
</rrl>
```

12.3.9 <*dnssec-policy*> section

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the dnssec-policy section.
description	STR	-	Description for the dnssec-policy section.
key-suite	STR	-	**id** of the <*key-suite*> to be used.
denial	STR	nsec	**id** of the <*denial*> to be used for *NSEC3* or the argument 'nsec' to use *NSEC*.

DNSSEC-POLICY SECTION

configuration example with <*denial*>

```
<dnssec-policy>
    id                  "dnssec-policy-nsec3"

    description         "Example of ZSK and KSK"
    denial              "nsec3-resalting-on"
    key-suite           "zsk-1024"
    key-suite           "ksk-2048"
</dnssec-policy>
```

configuration example without *<denial>*

```
<dnssec-policy>
    id                  "dp-nsec"

    description         "Example of ZSK and KSK"
    denial              "nsec"
    key-suite           "zsk-1024"
    key-suite           "ksk-2048"
</dnssec-policy>
```

12.3.10 *<key-suite>* section

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the key-suite section.
key-template	STR	-	**id** of the *<key-template>* to be used.
key-roll	STR	-	**id** of the *<key-roll>* to be used.

KEY-SUITE SECTION

configuration example *<key-suite>*

```
<key-suite>
    id                  "ksk-2048"

    key-template        "ksk-2048"
    key-roll            "key-roll-ksk-2048-short-times"
</key-suite>
```

12.3.11 <*key-roll*> section

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the key-roll section.
generate; generated; create	STR	-	Time when the key must be generated.
publish	STR	-	Time when the key must be published in the zone.
activate	STR	-	Time when the key will be used for signing the zone or apex of the zone.
inactive	STR	-	Time when the key will not be used anymore for signing.
delete	STR	-	Time when the key will be removed out of the zone.

KEY-ROLL SECTION

configuration example section-key-roll

```
<key-roll>
    id                  "key-roll-ksk-2048"

#   command         minutes   hours     day       month     day-week  week
    create          0         8         4         2         *         *
    publish         0         12        4         2         *         *
    activate        0         12        14        2         *         *
    inactive        0         8         4         3         *         *
    delete          0         12        11        3         *         *
</key-roll>
```

12.3.12 <*key-template*> section

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the key-template section.
ksk	FLAG	false	When this flag is enabled a *KSK* will be generated. When disabled a *ZSK* will be generated.
algorithm	ENUM	7	Sets the algorithm of the key. Supported values are: DSA; 3; RSASHA1; 5; NSEC3DSA; 6; NSEC3RSASHA1; 7; RSASHA256; 8; RSASHA512; 10; ECDSAP256SHA256; 13; ECDSAP384SHA384; 14.
size	INT	0	The length of the key in bits (incompatible sizes will be rejected). (MIN: 0; MAX: 4096)

KEY-TEMPLATE SECTION

configuration example section-key-template

```
<key-template>
    id                  "ksk-2048"
    ksk                 true
    algorithm           8
    size                2048
    engine              default
</key-template>
```

12.3.13 <*denial*> section

PARAMETER	TYPE	DEFAULT	DESCRIPTION
id	STR	-	**id** of the denial section.
salt	HEXSTR	-	The actual salt to use. Mutually exclusive with the **salt-length** option.
salt-length	INT	0	The system will generate a random salt with this length. Mutually exclusive with the **salt** option. (MIN: 0; MAX: 256)
iterations	INT	1	The number of iterations the salt and hash should be applied to the label. (MIN: 0; MAX: 65535)
optout	FLAG	false	When this flag is enabled only delegations which have a *DS* record will be considered for *NSEC3* record generation.

DENIAL SECTION

configuration example <*denial*>

```
<denial>
    id                  "nsec3-resalting-on"

    salt                "ABCD"
    #salt-length        4
    iterations          5
    optout              off
</denial>
```

13 Zones

Only textual zones are implemented.

The format of a zone file is defined in **RFC 1034**[43] and **RFC 1035**[44].

zone file sample

```
;; Example domain
$TTL    86400   ; 24 hours
$ORIGIN somedomain.eu.

somedomain.eu.      86400   IN  SOA ns1.somedomain.eu.  info.somedomain.eu. (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )

                    86400   IN  MX  10 mail.somedomain.eu.
                    86400   IN  NS  ns1.somedomain.eu.

ns1.somedomain.eu.  86400   IN  A   192.0.2.2
mail.somedomain.eu. 86400   IN  A   192.0.2.3
www.somedomain.eu.  86400   IN  A   192.0.2.4
```

13.1 MACROS

Some macros are implemented:

- @
- $TTL

- $ORIGIN

13.1.1 @

Use as a name, the @ symbol is replaced by the current origin.
The initial value is the **domain** field of the <zone> section.

For example:

configuration sample

```
<zone>
    domain somedomain.eu
    ...
</zone>
```

zone file sample

```
;; The following @ is seen as somedomain.eu.

@                   86400   IN  SOA ns1.somedomain.eu.  info.somedomain.eu. (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )
```

13.1.2 $TTL

This macro is the **TTL** value that is to be set for the resource records with an undefined **TTL**.

zone file sample

```
;; The following @ is seen as somedomain.eu.

$TTL 3600

somedomain.eu.        86400   IN  SOA ns1.somedomain.eu.  info.somedomain.eu. (
                                      1
                                      3600
                                      1800s
                                      3600000s
                                      600
                                      )
ns1.somedomain.eu.    86400 A   192.0.2.2

mail.somedomain.eu.   86400 A   192.0.2.3

www.somedomain.eu.    86400 A   192.0.2.4
                            A   192.0.2.5

ftp.somedomain.eu.          A   192.0.2.6 ;; The TTL will be set using $TTL
```

13.1.3 $ORIGIN

The value of this macro is appended to any following domain name not terminating with a ".". The initial value is the **domain** field of the <zone> section.

zone file sample

```
;; The following @ is seen as somedomain.eu.

$TTL 3600
$ORIGIN somedomain.eu.
somedomain.eu.              86400   IN  SOA ns1 info (
                                      1
                                      3600
                                      1800s
                                      3600000s
                                      600
                                      )
ns1                         86400 A   192.0.2.2
mail                        86400 A   192.0.2.3
www                         86400 A   192.0.2.4
```

13.2 Classes

YADIFA knows only one class:

- IN [44].

13.3 Resource record types

As master name server, *YADIFA* knows only the following *RR* types. Everything else will give an error and be ignored.

TYPE	VALUE	REFERENCE	SUPPORTED
A	1	**RFC 1035**[44]	Y
NS	2	**RFC 1035**[44]	Y
MD	3	**RFC 1035**[44]	N
MF	4	**RFC 1035**[44]	N
CNAME	5	**RFC 1035**[44]	Y
SOA	6	**RFC 1035**[44]	Y
MB	7	**RFC 1035**[44]	N
MG	8	**RFC 1035**[44]	N
MR	9	**RFC 1035**[44]	N
NULL	10	**RFC 1035**[44]	N
WKS	11	**RFC 1035**[44]	Y
PTR	12	**RFC 1035**[44]	Y
HINFO	13	**RFC 1035**[44]	Y
MINFO	14	**RFC 1035**[44]	N
MX	15	**RFC 1035**[44]	Y
TXT	16	**RFC 1035**[44]	Y
RP	17	**RFC 1183**[55]	N
AFSDB	18	**RFC 1183**[55] **RFC 5864**[7]	N
X25	19	**RFC 1183**[55]	N
ISDN	20	**RFC 1183**[55]	N
RT	21	**RFC 1183**[55]	N
NSAP	22	**RFC 1706**[17]	N
NSAP-PTR	23	**RFC 1348**[16] **RFC 1637**[18] **RFC 1706**[17]	N
SIG	24	**RFC 4034**[50] **RFC 3755**[59] **RFC 2535**[23] **RFC 2536**[24] **RFC 2537**[25] **RFC 2931**[2] **RFC 3110**[3] **RFC 3008**[60]	N

SUPPORTED TYPES

TYPE	VALUE	REFERENCE	SUPPORTED
KEY	25	**RFC 4034**[50] **RFC 3755**[59] **RFC 2535**[23] **RFC 2536**[24] **RFC 2537**[25] **RFC 2539**[26] **RFC 3008**[60] **RFC 3110**[3]	N
PX	26	**RFC 2163**[8]	N
GPOS	27	**RFC 1712**[11]	N
AAAA	28	**RFC 3596**[53]	Y
LOC	29	**RFC 1876**[21]	N
NXT	30	**RFC 3755**[59] **RFC 2535**[23]	N
EID	31	DNS Resource Records for Nimrod Routing Architecture	N
NIMLOC	32	DNS Resource Records for Nimrod Routing Architecture	N
SRV	33	**RFC 2782**[27]	Y
ATMA	34	ATM Name System V2.0	N
NAPTR	35	**RFC 2915**[20] **RFC 2168**[42] **RFC 3403**[41]	Y
KX	36	**RFC 2230**[9]	N
CERT	37	**RFC 4398**[37]	N
A6	38	**RFC 3226**[29] **RFC 2874**[34] **RFC 6563**[15]	N
DNAME	39	**RFC 6672**[61]	N
SINK	40	The Kitchen Sink Resource Record	N
OPT	41	**RFC 6891**[56] **RFC 3225**[?]	N
APL	42	**RFC 3123**[39]	N
DS	43	**RFC 4034**[50] **RFC 3658**[30]	Y
SSHFP	44	**RFC 4255**[28]	Y
IPSECKEY	45	**RFC 4025**[48]	N
RRSIG	46	**RFC 4034**[50] **RFC 3755**[59]	Y
NSEC	47	**RFC 4034**[50] **RFC 3755**[59]	Y
DNSKEY	48	**RFC 4034**[50] **RFC 3755**[59]	Y
DHCID	49	**RFC 4701**[31]	N
NSEC3	50	**RFC 5155**[13]	Y
NSEC3PARAM	51	**RFC 5155**[13]	Y
TLSA	52	**RFC 6698**[52]	Y
HIP	55	**RFC 5205**[40]	N
NINFO	56	The Zone Status (ZS) DNS Resource Record	N
RKEY	57	ENUM Encryption	N
TALINK	58	talink-completed-template	N
CDS	59	**RFC 7344**[12]	N
CDNSKEY	60	**RFC 7344**[12]	N
OPENPGPKEY	61	Using DANE to Associate OpenPGP public keys with email addresses	N
CSYNC	62	**RFC 7477**[33]	N

SUPPORTED TYPES

TYPE	VALUE	REFERENCE	SUPPORTED
SPF	99	**RFC 7208**[38]	N
UINFO	100	[IANA-Reserved]	N
UID	101	[IANA-Reserved]	N
GID	102	[IANA-Reserved]	N
UNSPEC	103	[IANA-Reserved]	N
NID	104	**RFC 6742**[51]	N
L32	105	**RFC 6742**[51]	N
L64	106	**RFC 6742**[51]	N
LP	107	**RFC 6742**[51]	N
EUI48	108	**RFC 7043**[5]	N
EUI64	109	**RFC 7043**[5]	N
TKEY	249	**RFC 2930**[19]	N
TSIG	250	**RFC 2845**[1]	N
IXFR	251	**RFC 1995**[45]	N
AXFR	252	**RFC 1035**[44] **RFC 5936**[22]	N
MAILB	253	**RFC 1035**[44]	N
MAILA	254	**RFC 1035**[44]	N
*	255	**RFC 1035**[44] **RFC 6895**[4]	N
URI	256	The Uniform Resource Identifier (URI) DNS Resource Record	N
CAA	257	**RFC 6844**[54]	N
TA	32768	Deploying DNSSEC Without a Signed Root	N
DLV	32769	**RFC 4431**[58]	N

SUPPORTED TYPES

14 Journal

YADIFA has got a new journaling system since the release of version 2.1.0.

The old system:

- is based on a append-only file
- has a linear access time (with the exception of the last few entries) which was not ideal for random access on big journals
- could only be limited in growth by emptying it completely

The new system:

- is based on a file that is being written in a cyclic fashion
- has a relatively constant access time
- can be limited in size, although it is not a hard limit.

The journal size is automatically set by *YADIFA* at around half the size of the zone size, but it can be set to an arbitrary value through configuration. To do this, one merely needs to set journal-size-kb in the <*zone*> section of the zone. The value range for version 2.1.0 is 64KB to maximum 8GB.

configuration example

```
<zone>
    domain somedomain.eu
    ...
    journal-size-kb 64
</zone>
<zone>
    domain someotherdomain.eu
    ...
    journal-size-kb 256000
</zone>
```

Note that this size is a soft limit. In several cases, *YADIFA* will exceed that value.

- Incremental updates have to be written completely. One incremental change could in theory have a wire size of up to about 64KB, which may result in as many exeeding bytes.
- When closing, the journal may write an index table with a size relative to the journal's size. From a 24 bytes for a small journal to a few megabytes for gigabytes-sized journals.

In order to reduce the size of the journal after reconfiguring it, it is recommended that one uses the command line to synchonize the zone and wipe the journal empty.

15 Statistics

YADIFA has a range of statistics available with one configuration setting. The statistics logger values are grouped into inputs, outputs and the RRL. Groups are composed of a name followed by an open parenthesis containing several space-separated event=count fields and ending in a closed parenthesis.

A single line of statistics looks as follows:

```
shell
udp (in=303 qr=303 ni=0 up=0 dr=0 st=91191 un=0 rf=0) tcp (in=369 qr=368 ni=0 up=0 dr=0 st=82477 un=0
rf=0 ax=0 ix=0 ov=0) udpa (OK=242 FE=0 SF=0 NE=0 NI=0 RE=61 XD=0 XR=0 NR=0 NA=0 NZ=0 BV=0 BS=0 BK=0
BT=0 BM=0 BN=0 BA=0 TR=0) tcpa (OK=209 FE=0 SF=0 NE=0 NI=0 RE=159 XD=0 XR=0 NR=0 NA=0 NZ=0 BV=0 BS=0
BK=0 BT=0 BM=0 BN=0 BA=0 TR=0) rrl (sl=0 dr=0)
```

You can clearly see the groups containing the event=count fields. There are currently 5 groups defined:

- udp(...) covers the UDP messages
- udpa(...) covers the UDP messages answers
- tcp(...) covers the TCP messages
- tcpa(...) covers the TCP messages answers
- rrl(...) covers the RRL events

The statistics logger counts the various events about the messages from the clients.

in input count
: counts the number of *DNS* messages received

qr query count
: counts the number of queries among the *DNS* messages

ni notify count
counts the number of notifications among the *DNS* messages

up update count
counts the number of updates among the *DNS* messages

dr dropped count
counts the number of *DNS* messages dropped

st total bytes sent (simple queries only)
counts the total number of bytes sent

un undefined opcode count
counts the number of undefined opcodes among the *DNS* messages

rf referral count
counts the number of referrals among the DNS queries

ax *AXFR* query count (TCP only)
counts the number of full zone transfers queried

ix *IXFR* query count (TCP only)
counts the number of incremental zone transfers queried

ov connection overflow (TCP only)
counts the number of times the TCP pool has been full when a new connection came in

The statistics logger answers counts the status of *DNS* answers sent to the clients.

OK NOERROR answer count

FE FORMERR answer count

SF SERVFAIL answer count

NE NXDOMAIN answer count

NI NOTIMP answer count

RE REFUSED answer count

XD YXDOMAIN answer count

XR YXRRSET answer count

NR NXRRSET answer count

NA NOTAUTH answer count

NZ NOTZONE answer count

BV BADVERS answer count

BS BADSIG answer count

BK BADKEY answer count

BT BADTIME answer count

BM BADMODE answer count

BN BADNAME answer count

BA BADALG answer count

TR BADTRUNC answer count

The RRL group only counts the two main events of the Response Rate Limiter.

dr dropped answer count
: counts the number of times an answer has been dropped

sl truncated answer count
: counts the number of times an answer that should have been dropped has been sent truncated instead

16 Configuration Examples

16.1 Introduction

zone file sample

```
;; Example domain
$TTL    86400   ; 24 hours
$ORIGIN somedomain.eu.

somedomain.eu.      86400   IN  SOA ns1.somedomain.eu.  info.somedomain.eu. (
                                    1
                                    3600
                                    1800s
                                    3600000s
                                    600
                                    )

                    86400   IN  MX  10 mail.somedomain.eu.
                    86400   IN  NS  ns1.somedomain.eu.

ns1.somedomain.eu.  86400   IN  A   192.0.2.2
mail.somedomain.eu. 86400   IN  A   192.0.2.3
www.somedomain.eu.  86400   IN  A   192.0.2.4
```

16.2 YADIFA as a primary name server

16.2.1 The One That is Really Easy

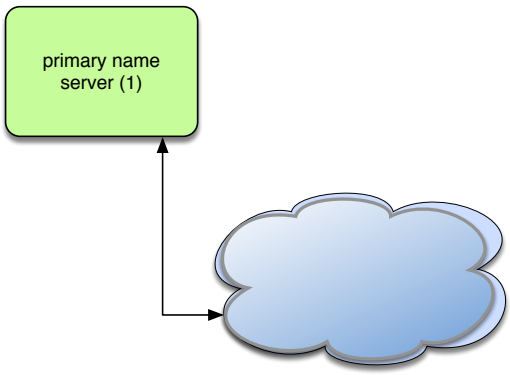

Figure 16.1: Primary name server (simple configuration)

configuration example of That is Really Easy

```
<zone>
    domain              somedomain.eu
    file                "masters/somedomain.eu."
    type                "master"
</zone>
```

16.2.2 The One With Activation of Logging

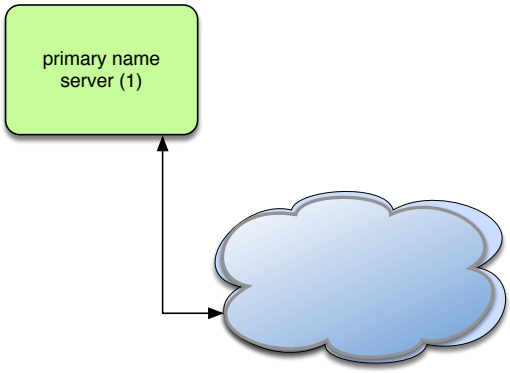

Figure 16.2: Primary name server with logging

configuration example of Activation of Logging

```
<channels>
    # user-defined-name     parameters
    # channel 'statistics': a file called stats.log
    #                       with 0644 access rights
    #
    statistics              stats.log 0644

    # channel 'syslog'    : a syslog daemon output using
    # the local6 facility and logging the pid of the process
    #
    syslog                  syslog local6,pid

    # channel 'yadifa'    : a file called yadifa.log with 0644 access rights
    #
    yadifa                  yadifa.log 0644

    # channel 'debug-out' : directly printing to stdout
    #
    debug-out               STDOUT

    # channel 'debug-err' : directly printint to stderr
    #
    debug-err               STDERR
</channels>

<loggers>
    # info, notice and warning level messages from the database logging
    # will be output
    database        info,notice,warning     yadifa
    database        err,crit,alert,emerg    yadifa,syslog
    server          *                       yadifa
    stats           *                       statistics
    system          *                       debug-err
    queries         *                       queries
    zone            *                       yadifa
</loggers>

<zone>
    domain              somedomain.eu
    file                "masters/somedomain.eu."
    type                "master"
</zone>
```

16.2.3 The One With NSID

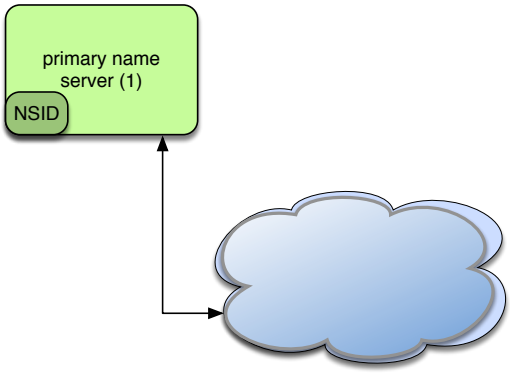

Figure 16.3: Primary name server with NSID

configuration example of NSID

```
<nsid>
    ascii "yadifad example NSID"
    # alternatively, an hexadecimal format can be used
    # hex 796164696661642065786d706c65204e5349440a
</nsid>

<zone>
    domain              somedomain.eu
    file                "masters/somedomain.eu."
    type                "master"
</zone>
```

16.2.4 The One With RRL

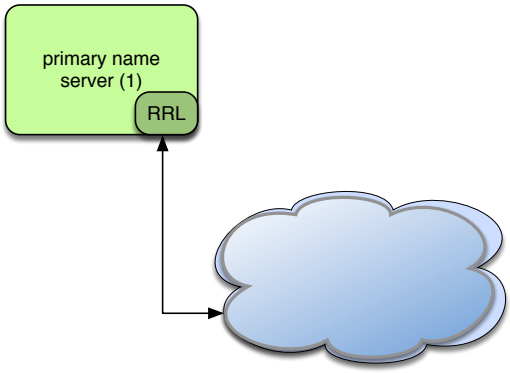

Figure 16.4: Primary name server with RRL

configuration example of RRL

```
# If YADIFA has been compiled with the Response Rate Limiter (default)
<rrl>
    # enable the RRL
    enabled             true

    # don't actually limit the response rate, only log what the filter
    # would do
    log-only            false

    # how many responses per second are allowed for a client
    # (masked with the prefix)
    responses-per-second 5

    # how many errors per second are allowed for a client
    # (masked with the prefix)
    errors-per-second   5

    # window of time in which the rates are measured, expressed in seconds
    window              15

    # every "slip" dropped answers, a truncated answer may randomly be
    # given so the client can ask again using TCP
    slip                2

    # the min size of the table storing clients(masked with the prefix)
    min-table-size      1024

    # the max size of the table storing clients(masked with the prefix)
    max-table-size      16384

    # IPv4 clients are masked with this prefix
    ipv4-prefix-length  24

    # IPv6 clients are masked with this prefix
    ipv6-prefix-length  56

    # the list of IP/networks (Access Control List) not impacted by
    # the RRL
    exempted            none
</rrl>

<zone>
    domain              somedomain.eu
    file                "masters/somedomain.eu."
    type                "master"
</zone>
```

16.2.5 The One With DNSSEC Policy 'diary' style

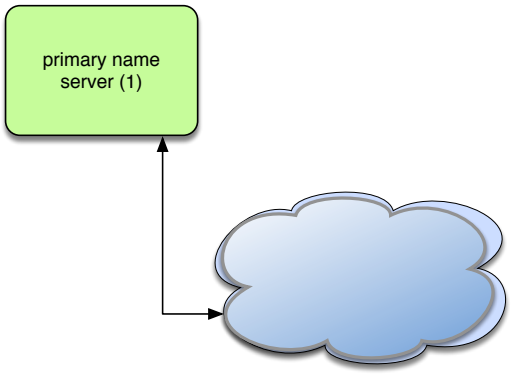

Figure 16.5: Primary name server (DNSSEC policy 'diary' style)

configuration example of DNSSEC Policy 'diary' style

```
<key-roll>
    id                  "key-roll-ksk-2048"

#   command         minutes   hours     day       month     day-week  week
    create          0         8         4         2         *         *
    publish         0         12        4         2         *         *
    activate        0         12        14        2         *         *
    inactive        0         8         4         3         *         *
    delete          0         12        11        3         *         *
</key-roll>

<key-roll>
    id                  "key-roll-zsk-1024"

#   command         minutes   hours     day       month     day-week  week
    create          0         10        5         *         *         *
    publish         0         12        5         *         *         *
    activate        0         14        5         *         *         *
    inactive        0         10        6         *         *         *
    delete          0         10        11        *         *         *
</key-roll>

<key-suite>
    id                  "ksk-2048"
    key-template        "ksk-2048"
    key-roll            "key-roll-ksk-2048"
</key-suite>

<key-suite>
    id                  "zsk-1024"
```

```
    key-template        "zsk-1024"
    key-roll            "key-roll-zsk-1024"
</key-suite>

<dnssec-policy>
    id                  "dp-nsec"

    description         "Example of ZSK and KSK"
    denial              "nsec"
    key-suite           "zsk-1024"
    key-suite           "ksk-2048"
</dnssec-policy>

<zone>
    domain              somedomain.eu
    file                masters/somedomain.eu.
    type                "master"
    dnssec-policy       "dp-nsec"
</zone>
```

16.2.6 The One With DNSSEC Policy 'relative' style

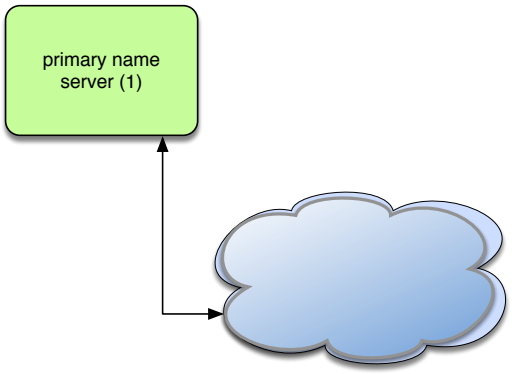

Figure 16.6: Primary name server (DNSSEC policy 'relative' style)

configuration example of DNSSEC Policy 'relative' style

```
<key-roll>
    id                  "key-roll-ksk-2048"

    create              +355d
    publish             +4h
    activate            +10d
    inactive            +366d
    delete              +7d

</key-roll>

<key-roll>
    id                  "key-roll-zsk-1024"

    create              +30d
    publish             +2h
    activate            +7200 # 2 hours (in seconds)
    inactive            +31d
    delete              +7d

</key-roll>

<key-suite>
    id                  "ksk-2048"
    key-template        "ksk-2048"
    key-roll            "key-roll-ksk-2048"
</key-suite>

<key-template>
    id                  "ksk-2048"
```

```
    ksk                 true
    algorithm           8
    size                2048
    engine              default
</key-template>

<key-suite>
    id                  "zsk-1024"
    key-template        "zsk-1024"
    key-roll            "key-roll-zsk-1024"
</key-suite>

<key-template>
    id                  "zsk-1024"
    algorithm           8
    size                1024
    engine              default
</key-template>

<denial>
    id                  "nsec3-resalting-on"

    salt                "ABCD"
    #salt-length        4
    iterations          5
    optout              off
</denial>

<dnssec-policy>
    id                  "dnssec-policy-nsec3"

    description         "Example of ZSK and KSK"
    denial              "nsec3-resalting-on"
    key-suite           "zsk-1024"
    key-suite           "ksk-2048"
</dnssec-policy>

<zone>
    domain              somedomain.eu
    file                masters/somedomain.eu.
    type                "master"
    dnssec-policy       "dnssec-policy-nsec3"
</zone>
```

16.2.7 The One With the Controller

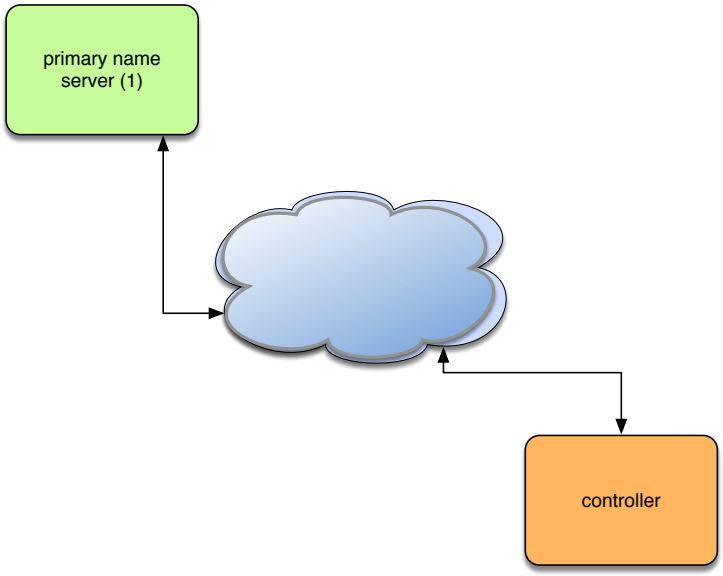

Figure 16.7: Primary name server with controller

On the primary name server (SYSCONFDIR/yadifad.conf):

configuration example of Controller (server)

```
<main>
    allow-control       "yadifa-controller"
</main>

<acl>
    yadifa-controller   key "controller-key"
</acl>

<key>
    name                "controller-key"
    algorithm           "hmac-md5"
    secret              "ControlDaemonKey"
</key>
```

On the controller (${HOME}/.yadifa.rc or SYSCONFDIR/yadifa.conf):

configuration example of Controller (client)

```
<yadifa>
    server              192.0.2.1
    tsig-key-name       "controller-key"
</yadifa>

<key>
    name                "controller-key"
    algorithm           "hmac-md5"
    secret              "ControlDaemonKey"
</key>
```

16.3 YADIFA as a secondary name server

16.3.1 The One With One Master

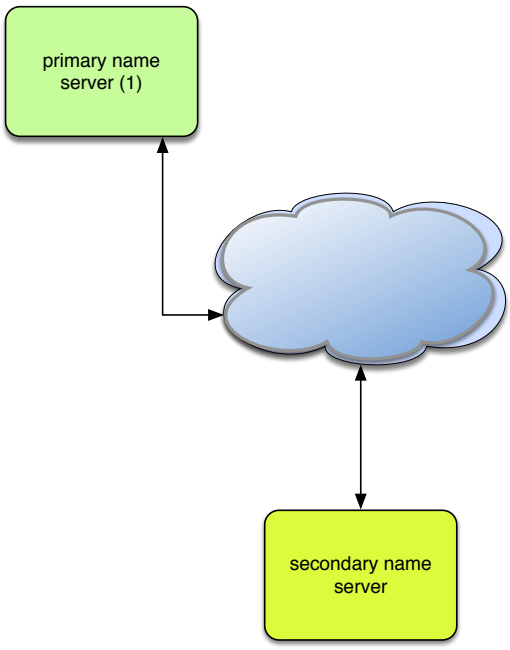

Figure 16.8: Secondary name server (one master)

configuration example of One Master

```
<zone>
    domain              somedomain.eu
    file                "slaves/somedomain.eu."
    type                "slave"
    master              192.0.2.1
</zone>
```

16.3.2 The One With Several Masters

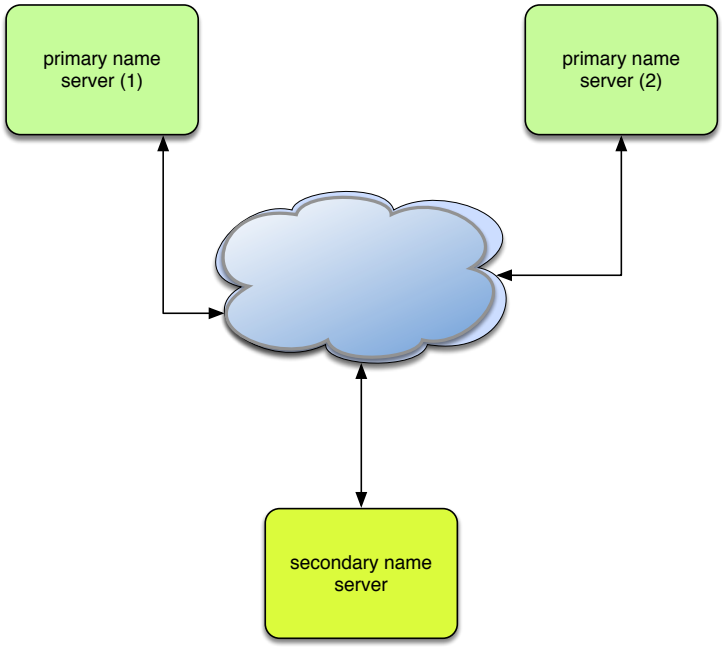

Figure 16.9: Secondary name server (several masters)

configuration example of Several Masters

```
<zone>
    domain              somedomain.eu
    file                "slaves/somedomain.eu."
    type                "slave"

    masters             192.0.2.1,192.0.2.2
    true-multimaster    yes
</zone>
```

16.3.3 The One With Activation of Logging

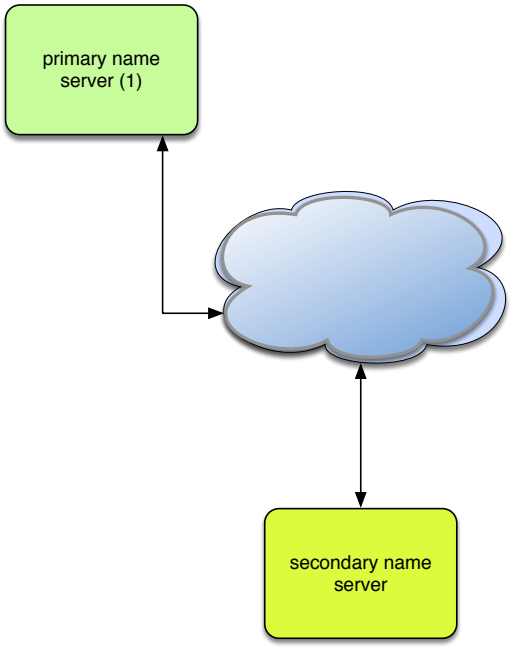

Figure 16.10: Secondary name server with logging

configuration example of Activation of Loggin

```
<channels>
    # user-defined-name     parameters
    # channel 'statistics': a file called stats.log
    #                       with 0644 access rights
    #
    statistics              stats.log 0644

    # channel 'syslog'    : a syslog daemon output using
    # the local6 facility and logging the pid of the process
    #
    syslog                  syslog local6,pid

    # channel 'yadifa'    : a file called yadifa.log with 0644 access rights
    #
    yadifa                  yadifa.log 0644

    # channel 'debug-out' : directly printing to stdout
    #
    debug-out               STDOUT

    # channel 'debug-err' : directly printint to stderr
    #
    debug-err               STDERR
</channels>

<loggers>
    # info, notice and warning level messages from the database logging
    # will be output
    database        info,notice,warning     yadifa
    database        err,crit,alert,emerg    yadifa,syslog
    server          *                       yadifa
    stats           *                       statistics
    system          *                       debug-err
    queries         *                       queries
    zone            *                       yadifa
</loggers>

<zone>
    domain              somedomain.eu
    file                "slaves/somedomain.eu."
    type                "slave"
    master              192.0.2.1
</zone>
```

16.3.4 The One With NSID

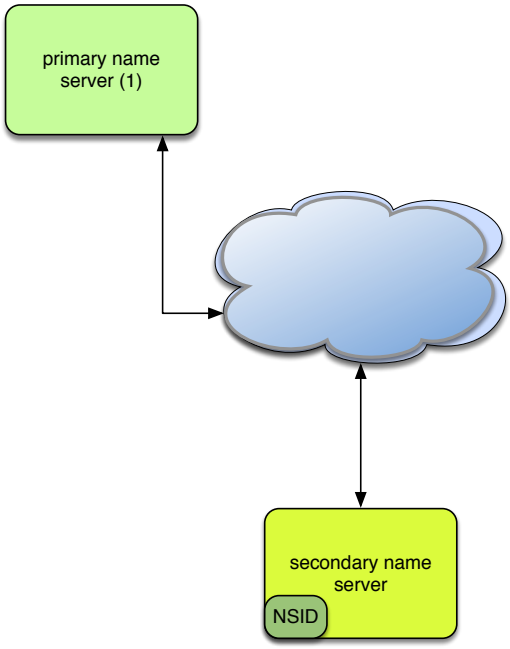

Figure 16.11: Secondary name server with NSID

configuration example of NSID

```
<nsid>
    ascii "yadifad example NSID"
    # alternatively, an hexadecimal format can be used
    # hex 796164696661642065786d706c65204e5349440a
</nsid>

<zone>
    domain              somedomain.eu
    file                "slaves/somedomain.eu."
    type                "slave"
    master              192.0.2.1
</zone>
```

16.3.5 The One With RRL

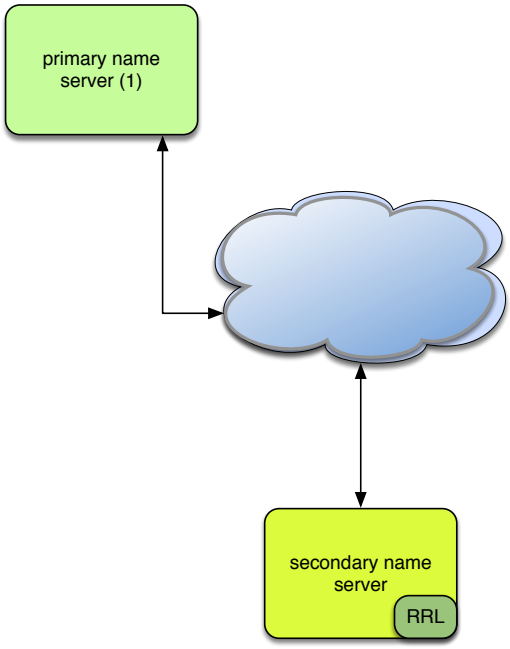

Figure 16.12: Secondary name server with RRL

configuration example of RRL

```
# If YADIFA has been compiled with the Response Rate Limiter (default)
<rrl>
    # enable the RRL
    enabled             true

    # don't actually limit the response rate, only log what the filter
    # would do
    log-only            false

    # how many responses per second are allowed for a client
    # (masked with the prefix)
    responses-per-second 5

    # how many errors per second are allowed for a client
    # (masked with the prefix)
    errors-per-second   5

    # window of time in which the rates are measured, expressed in seconds
    window              15

    # every "slip" dropped answers, a truncated answer may randomly be
    # given so the client can ask again using TCP
    slip                2

    # the min size of the table storing clients(masked with the prefix)
    min-table-size      1024

    # the max size of the table storing clients(masked with the prefix)
    max-table-size      16384

    # IPv4 clients are masked with this prefix
    ipv4-prefix-length  24

    # IPv6 clients are masked with this prefix
    ipv6-prefix-length  56

    # the list of IP/networks (Access Control List) not impacted by
    # the RRL
    exempted            none
</rrl>

<zone>
    domain              somedomain.eu
    file                "slaves/somedomain.eu."
    type                "slave"
    master              192.0.2.1
</zone>
```

17 Troubleshooting

By default *YADIFA* logs everything on the standard output. Warnings or errors may point to the issue. When configuring the logging to suit your needs, it is recommended one keeps the levels: warning,err,crit,alert and emerg for everything but the queries.

17.1 Submitting a bug report

If you are unable to fix the issue yourself, you can submit a bug report to the *YADIFA* team.
For critical issues (i.e.: crash), please use bugreport@yadifa.eu.
For any other issue or question, you can use yadifa-users@mailinglists.yadifa.eu.

The report should contain:

- The operating system type and version
- The version of *YADIFA* and how it was installed.
  - If you configured it yourself : the *./configure* parameters
  - If you used a package : where from and what version
- What machine it is running on
- All the log output, preferrably with all levels enabled (* or any in the configuration file).
- If you know them: the steps to reproduce the issue
- If possible, the zone files and as much of the configuration file you can give (i.e.: everything but the TSIG keys)

Please find enclosed two short scripts you can run on the server to retrieve most of the information we need.

System information (some programs or files will not exist on your system):

script

```
#!/bin/sh

# basic system information
echo uname:
echo ------
uname -a
# OS
cat /etc/lsb_release
cat /etc/redhat-release
cat /etc/slackware-version
cat /etc/os-release
cat /etc/defaults/pcbsd
cat /etc/defaults/trueos
echo mount:
echo ------
mount
# available disk space
echo df:
echo ---
df -h

# available memory space
echo free:
echo -----
free -h
```

Hardware information:

script

```
#!/bin/sh
# various hardware information

echo lscpu:
echo ------
lscpu

echo lspci:
echo ------
lspci

echo lshw:
echo -----
lshw

echo hwinfo:
echo -------
hwinfo

echo lsscsi:
echo -------
lsscsi

echo lsusb:
echo ------
lsusb

echo lsblk:
echo ------
lsblk

echo pciconf:
echo --------
pciconf -lvcb
```

Please find enclosed a short script you can run on the build machine to retrieve information about the compiler:

script

```
#/bin/sh

# compiler info (if you compiled yadifad yourself)
# to run on the build machine

echo gcc:
echo ----
gcc -v -v
gcc -dM -E - < /dev/null

echo clang:
echo ------
clang -v -v
clang -dM -E - < /dev/null
```

17.2 Stacktrace

In the case of a crash, generating a stacktrace at the time of the problem arises may help to understand the issue. Please note that it is best to do this with the debug symbols for the package installed or with a binary that has not been stripped.

To generate the stacktrace, you can either use a generated core dump, or run yadifad in the debugger.

Please note that the way to enable unlimited-size core dumps varies with your OS flavor. On some linux, you can get its location by executing:

shell

```
$ cat /proc/sys/kernel/core_pattern
```

And enable it typing, as root:

shell

```
$ ulimit -c unlimited
```

Be sure the command worked:

shell

```
$ ulimit -c
```

Should print:

shell output

```
unlimited
```

17.2.1 Using a core dump

With a core dump at hand, you can start the debugger like this:

gdb */path-to-yadifad/yadifad /path-to-yadifad-core-dump/yadifad-core-dump-file*

For example:

shell

```
$ gdb /usr/local/sbin/yadifad /var/cache/abrt/yadifad.core
```

Then on the debugger prompt:

gdb

```
set logging file /tmp/yadifad-stacktrace.txt
set logging on
thread apply all bt
```

You can keep pressing the [enter] key until you are back to an empty (gdb) prompt

gdb

```
quit
```

The file /tmp/yadifad-stacktrace.txt will contain the stacktraces.

17.2.2 Running yadifad in the debugger

You can start the debugger like this:

gdb */path-to-yadifad/yadifad*

```
shell
$ gdb /usr/local/sbin/yadifad
```

Or, if yadifad is already running, like this:

gdb -p *yadifad-pid*

```
shell
$ gdb -p 12345
```

Then on the debugger prompt:

```
gdb
handle SIGUSR1 noprint pass
handle SIGUSR2 noprint pass
handle SIGTERM noprint pass
handle SIGINT noprint pass
handle SIGPIPE noprint pass
handle SIGHUP noprint pass
handle SIG33 noprint pass
set follow-fork-mode child
run
```

When the debugger stops with an error (i.e.: SIGSEGV, SIGABRT):

```
gdb
set logging file /tmp/yadifad-stacktrace.txt
set logging on
thread apply all bt
```

You can keep pressing the [enter] key until you get an empty (gdb) prompt.

```
gdb
quit
```

The file /tmp/yadifad-stacktrace.txt will contain the stacktraces.

17.3 Building yadifad with even more debugging information

When preparing to build yadifad, there are *./configure* options that increase the debugging information available.

The stacktrace information in the logs can be improved using *–enable-bfd-debug*. The cost of this option can be considered negligible.
Please note that although very useful in some cases, the mutexes monitoring feature (enabled using *–enable-mutex-debug*) is extremely expensive and should only be used in very specific cases.

In order to enable more debugging information, the make target "debug" greatly increases logging and activates many runtime checks. All internal libraries must be compiled with the same target so start from a clean source.

```
shell
$ make clean
$ make debug
$ sudo make install
```

Note that this kind of build may generate extremely huge log files. The increased logging is still subject to the settings in yadifad.conf so it is still possible to tune the flow.

Bibliography

[1] Paul Vixie / O. Gudmundsson / D. Eastlake 3rd / B. Wellington. *Secret Key Transaction Authentication for DNS (TSIG)*, May 2000. **RFC 2845**.

[2] D. Eastlake 3rd. *DNS Request and Transaction Signatures (SIG(0)s)*, September 2000. **RFC 2931**.

[3] D. Eastlake 3rd. *RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)*, May 2001. **RFC 3110**.

[4] D. Eastlake 3rd. *Domain Name System (DNS) IANA Considerations*, April 2013. **RFC 6895**.

[5] J. Abley. *Resource Records for EUI-48 and EUI-64 Addresses in the DNS*, October 2013. **RFC 7043**.

[6] J. Abley. *Authenticated Denial of Existence in the DNS*, February 2014. **RFC 7129**.

[7] R. Allbery. *DNS SRV Resource Records for AFS*, April 2010. **RFC 5864**.

[8] C. Allocchio. *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)*, January 1998. **RFC 2163**.

[9] R. Atkinson. *Key Exchange Delegation Record for the DNS*, November 1997. **RFC 2230**.

[10] R. Austein. *DNS Name Server Identifier (NSID) Option*, August 2007. **RFC 5001**.

[11] C. Farrell / M. Schulze / S. Pleitner / D. Baldoni. *DNS Encoding of Geographical Location*, November 1994. **RFC 1712**.

[12] W. Kumari / O. Gudmundsson / G. Barwood. *Automating DNSSEC Delegation Trust Maintenance*, September 2014. **RFC 7344**.

[13] B. Laurie / G. Sisson / R. Arends / D. Blacka. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*, March 2008. **RFC 5155**.

[14] Paul Vixie / S. Thomson / Y. Rekhter / J. Bound. *Dynamic Updates in the Domain Name System (DNS UPDATE)*, April 1997. **RFC 2136**.

[15] S. Jiang / D. Conrad / B. Carpenter. *Moving A6 to Historic Status*, March 2012. **RFC 6563**.

[16] B. Manning / R. Colella. *DNS NSAP Resource Records*, July 1992. **RFC 1348**.

[17] B. Manning / R. Colella. *DNS NSAP Resource Records*, October 1994. **RFC 1706**.

[18] B. Manning / R. Colella. *DNS NSAP Resource Records*, June 1994. **RFC 1637**.

[19] 3rd D. Eastlake. *Secret Key Establishment for DNS (TKEY RR)*, September 2000. **RFC 2930**.

[20] M. Mealling / R. Daniel. *The Naming Authority Pointer (NAPTR) DNS Resource Record*, September 2000. **RFC 2915**.

[21] C. Davis / Paul Vixie / T. Goodwin / I. Dickinson. *A Means for Expressing Location Information in the Domain Name System*, January 1996. **RFC 1876**.

[22] Ed. E. Lewis / A. Hoenes. *DNS Zone Transfer Protocol (AXFR)*, June 2010. **RFC 5936**.

[23] D. Eastlake. *Domain Name System Security Extensions*, March 1999. **RFC 2535**.

[24] D. EastLake. *DSA KEYs and SIGs in the Domain Name System (DNS)*, March 1999. **RFC 2536**.

[25] D. Eastlake. *RSA/MD5 KEYs and SIGs in the Domain Name System (DNS)*, March 1999. **RFC 2537**.

[26] D. Eastlake. *Storage of Diffie-Hellman Keys in the Domain Name System (DNS)*, March 1999. **RFC 2539**.

[27] A. Gulbrandsen / Paul Vixie / L. Esibov. *A DNS RR for specifying the location of services (DNS SRV)*, February 2000. **RFC 2782**.

[28] J. Schlyter / W. Griffin. *Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints*, January 2006. **RFC 4255**.

[29] O. Gudmundsson. *DNSSEC and IPv6 A6 aware server/resolver message size requirements*, December 2001. **RFC 3226**.

[30] O. Gudmundsson. *Delegation Signer (DS) Resource Record (RR)*, December 2003. **RFC 3658**.

[31] M. Stapp / T. Lemon / A. Gustafsson. *DNS Resource Record (RR) for Encoding Dynamic Host Configuration Protocol (DHCP) Information (DHCID RR)*, October 2006. **RFC 4701**.

[32] S. Kwan / P. Garg / J. Gilroy / L. Esibov / J. Westhead / R. Hall. *Secret Key Transaction Authentication for DNS (GSS-TSIG)*, October 2003. **RFC 3645**.

[33] W. Hardaker. *Child-to-Parent Synchronization in DNS*, March 2015. **RFC 7477**.

[34] M. Crawford / C. Huitema. *DNS Extensions to Support IPv6 Address Aggregation and Renumbering*, July 2000. **RFC 2874**.

[35] J. Jansen. *Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC*, October 2009. **RFC 5702**.

[36] S. Josefsson. *Base-N Encodings*, October 2006. **RFC 4648**.

[37] S. Josefsson. *Storing Certificates in the Domain Name System (DNS)*, March 2006. **RFC 4398**.

[38] S. Kitterman. *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1*, April 2014. **RFC 7208**.

[39] P. Koch. *A DNS RR Type for Lists of Address Prefixes (APL RR)*, June 2001. **RFC 3123**.

[40] P. Nikander / J. Laganier. *Host Identity Protocol (HIP) Domain Name System (DNS) Extension*, April 2008. **RFC 5205**.

[41] M. Mealling. *Dynamic Delegation Discovery System (DDDS)*, October 2002. **RFC 3403**.

[42] R. Daniel / M. Mealling. *Resolution of Uniform Resource Identifiers using the Domain Name System*, June 1997. **RFC 2168**.

[43] Paul Mockapetris. *DOMAIN NAMES - CONCEPTS AND FACILITIES*, November 1987. **RFC 1034**.

[44] Paul Mockapetris. *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*, November 1987. **RFC 1035**.

[45] M. Ohta. *Incremental Zone Transfer in DNS*, August 1996. **RFC 1995**.

[46] John Postel. *USER DATAGRAM PROTOCOL*, Augustus 1980. **RFC 768**.

[47] John Postel. *TRANSMISSION CONTROL PROTOCOL*, September 1981. **RFC 793**.

[48] M. Richardson. *A Method for Storing IPsec Keying Material in DNS*, February 2005. **RFC 4025**.

[49] R. Arends / R. Austein / M. Larson / D. Massey / S. Rose. *DNS Security Introduction and Requirements*, March 2005. **RFC 4033**.

[50] R. Arends / R. Austein / M. Larson / D. Massey / S. Rose. *Resource Records for the DNS Security Extensions*, March 2005. **RFC 4034**.

[51] RJ Atkinson / SN Bhatti / S. Rose. *DNS Resource Records for the Identifier-Locator Network Protocol (ILNP)*, November 2012. **RFC 6742**.

[52] P. Hoffman / J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*, August 2012. **RFC 6698**.

[53] S. Thomson / C. Huitema / V. Ksinant / M. Souissi. *DNS Extensions to Support IP Version 6*, October 2003. **RFC 3596**.

[54] P. Hallam-Baker / R. Stradling. *DNS Certification Authority Authorization (CAA) Resource Record*, January 2013. **RFC 6844**.

[55] C. Everhart / L. Mamakos / R. Ullmann. *New DNS RR Definitions*, October 1990. **RFC 1183**.

[56] J. Damas / M. Graff / Paul Vixie. *Extension Mechanisms for DNS (EDNS(0))*, April 2013. **RFC 6891**.

[57] Paul Vixie. *Extension Mechanisms for DNS (EDNS0)*, August 1999. **RFC 2671**.

[58] M. Andrews / S. Weiler. *The DNSSEC Lookaside Validation (DLV) DNS Resource Record*, February 2006. **RFC 4431**.

[59] S. Weiler. *Legacy Resolver Compatibility for Delegation Signer (DS)*, May 2004. **RFC 3755**.

[60] B. Wellington. *Domain Name System Security (DNSSEC) Signing Authority*, November 2000. **RFC 3008**.

[61] S. Rose / W. Wijngaards. *DNAME Redirection in the DNS*, June 2012. **RFC 6672**.

Index