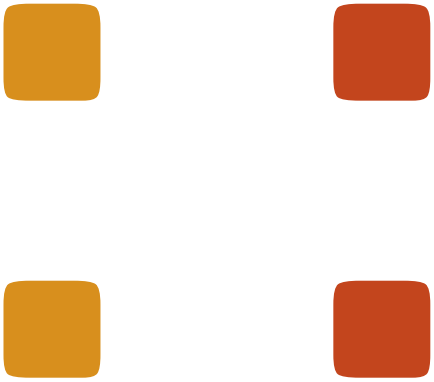

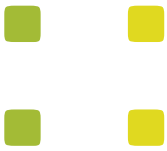

Reference manual

version 2.0.0

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

Yet Another DNS Implementation For All

Contents

List of Figures

List of Tables

1 Introduction

YADIFA is a *name server* implementation developed by **EUR***id* **vzw/absl**, the registry for the .eu top-level domain. **EUR***id* **vzw/absl** 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**[11]
- supports **DNSSEC** with **NSEC**[1] and **NSEC3**[2]
- has full and incremental zone transfer handling (**AXFR**[5] and **IXFR**[8]).

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
- Allow dynamic provisioning of zones without restart.

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

In future releases new features will be added:

- recursion
- caching
- validation
- split horizon
- plug-in system to integrate with **EURid** **vzw/absl**'s proprietary systems
- dynamic provisioning of new domain names
- **DNSSEC** signing service

1.1 Domain Name System

The Domain Name System (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*. Via the *root* server which guides you to the *eu* 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 **NS** resource record which contains the names of the *.eu* name servers. In the *.eu* zone information you will still not find

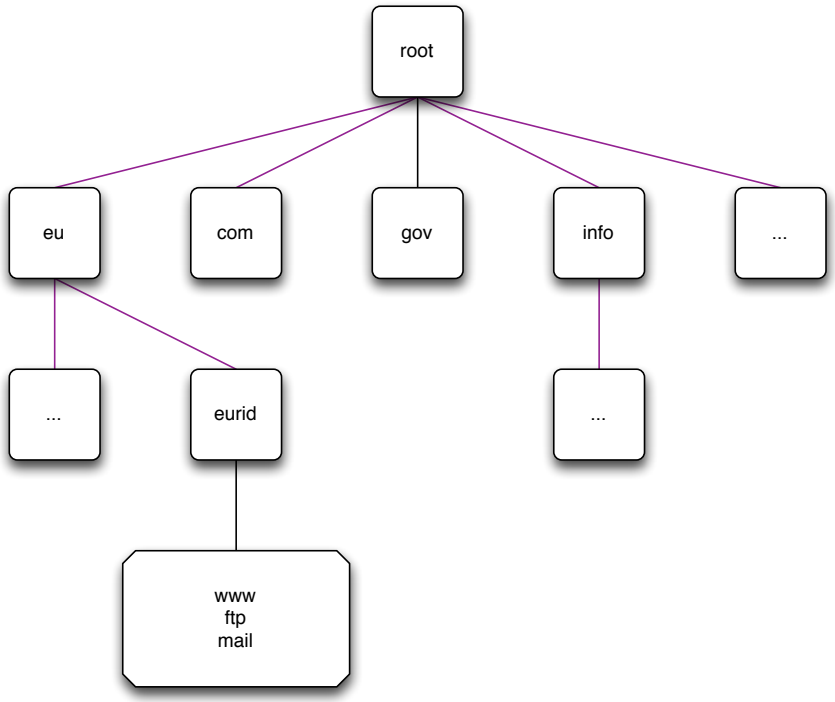

Figure 1.1: DNS hierarchy

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 in the zone information, including 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) resource record. 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

2.2 CPU

The 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.

2.3 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.

2.4 Supported Operating Systems

YADIFA has been compiled for x86, x86_64 on GNU/Linux (UBUNTU, Red Hat), FreeBSD and OSX. Other Unix flavours and Windows support are planned.

3 Installation

The current version of **YADIFA** is: 2.0.0

YADIFA is a collection of one daemon, *yadifad*; four libraries; two man pages, *yadifad.1* and *yadifad-conf.5*; and example configuration files.

The libraries are:

- dnscore
- dnsdb
- dnszone
- dnslg.

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

YADIFA is tested with:

- GCC 4.6
- CLANG 3.1-2
- ICC 12.1.3.

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

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

or

```
./configure CC=clang
```

or

```
./configure CC=icc
```

3.1 Server installation

YADIFA has several components:

- A daemon **yadifad**
- A man page yadifad.1
- A man page yadifad-conf.5
- A yadifad.conf.example file.

If we install *yadifa* in /opt/ we set the install_prefix to /opt/

```
install_prefix='/opt/'

tar zxvf yadifa-0.1.0-xxxx.tar.gz
cd yadifa-0.1.0-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/man1/
${install_prefix}/share/man/man5/
${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}/sbin/yadifad
${install_prefix}/share/man/man1/yadifad.1
${install_prefix}/share/man/man5/yadifad-conf.5
```

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/
```

3.2 Client installation

YADIFA has several components:

- A remote access tool **yadifa** for the server **yadifad**
- A name server lookup tool **yadifa**
- A man page for yadifa yadifa.1.

If we install *yadifad* in /opt/ we set the install_prefix to /opt/. For the client software you need to "configure" with *–with-tools*.

```
install_prefix='/opt/'

tar zxvf yadifa-0.1.0-xxxx.tar.gz
cd yadifa-0.1.0-xxxx

./configure --prefix=${install_prefix}/yadifa/ --with-tools
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/man1/
${install_prefix}/share/man/man5/
${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/man1/yadifa.1
${install_prefix}/share/man/man1/yadifad.1
${install_prefix}/share/man/man5/yadifad-conf.5
```

and the include files in:

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

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, it just needs 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 eight sections:

- main (see 10.2)
- zone (see 10.2)
- key (see 10.2)
- acl (see 10.2)
- channels (see 10.2)
- loggers (see 10.2)
- nsid (see 10.2)
- rrl (see 10.2).

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
- **channel and loggers** are needed to configure your log information

- **key** contains **TSIG**[4]information
- **nsid** contains the "DNS Name Server Identifier Option"
- **rrl** contains the "Response Rate Limiting in the Domain Name System".

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*.

For example:

```
<zone>
    domain          somedomain.eu
    file            masters/somedomain.eu.txt
    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 56.

4.1.2 Slave 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.

For example:

```
<zone>
    domain          somedomain.eu
    file            slaves/somedomain.eu.txt
    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
- **SIGHUP** will reopen the log files
- **SIGUSR1** will save all zone files to disk.

For example:

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

5 Server Technical

For now there are three entry points to the database:

1. Zone File
2. **AXFR**[5] and **IXFR**[8]
3. **DNS UPDATE**[10].

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**[10] just this package 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 11.3.

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 TCP connection. This communication can be authenticated with **TSIG**[4]'s. This **TSIG**[4] 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.

```
./configure --enable-ctrl
```

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

```
make
make install
```

Note:

> You also need to add 'allow-control' in the main section of *yadifad.conf*(10.2).

6.1.1 Commands

TYPES	ARGUMENTS
SHUTDOWN	
FREEZE	somedomain.eu
UNFREEZE	somedomain.eu
FREEZEALL	
UNFREEZEALL	
RELOAD	somedomain.eu
ZONECFGRELOAD	somedomain.eu

shutdown

This command shutdowns the server.

For example:

```
./yadifa -s 192.0.2.1 -t SHUTDOWN
```

Gives as result:

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

;; QUESTION SECTION:
;.                               CTRL    TYPE11009

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

freeze

This command suspends updates to a zone. No more modifcation (dyn DNS) can be done.

For example:

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

Gives as result:

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

;; QUESTION SECTION:
;.                               CTRL    TYPE11014

;; ANSWER SECTION:
.                        0       CTRL    TYPE11014  \# 15 A037F6D65646F6D61696
E620565700

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

unfreeze

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

For example:

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

Gives as result:

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

;; QUESTION SECTION:
;.                               CTRL    TYPE11015

;; ANSWER SECTION:
.                        0       CTRL    TYPE11015  \# 15 A037F6D65646F6D61696
E620565700

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

freezeall

This command suspends updates to all zones. No more modification (dyn DNS) can be done.

For example:

```
./yadifa -s 192.0.2.1 -t FREEZEALL
```

Gives as result:

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: ?, status: NOERROR, id: 49553
;; flags: qr QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                                CTRL    TYPE11014

;; Query time: 0 msec
;; WHEN: Mon Sep 29 14:57:22 2014
;; MSG SIZE rcvd: 17
```

unfreezeall

This command enables updates to all zones. Modifications (dyn DNS) can be done again.

For example:

```
./yadifa -s 192.0.2.1 -t UNFREEZEALL
```

Gives as result:

```
;; global options:
;; Got answer:
;; ->>HEADER<<- opcode: ?, status: NOERROR, id: 33527
;; flags: qr QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                               CTRL    TYPE11015

;; Query time: 0 msec
;; WHEN: Mon Sep 29 14:57:48 2014
;; MSG SIZE rcvd: 17
```

reload

This command reloads the zone file from disk.

For example:

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

Gives as result:

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

;; QUESTION SECTION:
;.                               CTRL    TYPE11018

;; ANSWER SECTION:
.                        0       CTRL    TYPE11018  \# 15 A037F6D65646F6D61696
E620565700

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

zonecfgreload

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

For example:

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

Gives as result:

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

;; QUESTION SECTION:
;.                               CTRL    TYPE11019

;; ANSWER SECTION:
.                        0       CTRL    TYPE11019  \# 15 A037F6D65646F6D61696
E620565700

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

7 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 (RRSIG) resource records. 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 its original one.

[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 (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 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 (DS) resource record.

There it is signed with the private part of the parent zone key pair. 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.

It is also understandable that TLD operators are working hard to publish their data with signatures, because it is only if a TLD is DNSSEC-enabled that receivers can find a completed chain of trust, allowing them to easily verify domain name signatures within that TLD. Now that the root zone is signed and TLDs sign their data as well, registrars are also able to sign their DNS data.

7.3 Types of key pairs

Two types of keys are used in DNSSEC:

- The key-signing key (KSK) - used only to sign the hash of DNSKEY information
- The zone -signing key (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 . RSASHA1-NSEC3 - SHA1 (algorithm number 7) is defined by RFC 5155 . 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.

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 NSID

8.1 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. 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.

8.2 NSID payload

The **NSID**[3] option as a sequence of hexadecimal digits, two digits per payload octet.

The payload of NSID is a maximum of 512 bytes long and can consist of any combination of bytes.

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.

9 DNS Response Rate Limiting

9.1 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.

9.2 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.

9.3 The problem

Any internet protocol based on 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.

9.4 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**.
- **Responses-per-second** is the maximum number of "no-error" answers that will be given to a client in the duration of a second.
- **Errors-per-second** is the maximum number of error answers that will be given to a client in the duration of a second.
- **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.

10 Configuration reference

10.0.1 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

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**.

For example:

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

For example:

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

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

10.1 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	DNSSEC type of the zone. Can be no-dnssec (none, no, off, 0), or dnssec (nsec, nsec3, nsec3-optout).
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	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)

Table 10.1: Types

10.2 Sections

The 'main' section

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
additional-from-auth	FLAG	true	If this flag is enabled, the server will reply with the additional section.
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	Dedault 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.
authority-from-auth	FLAG	true	If this flag is enabled, the server will reply with the authority section.
axfr-compress-packets	FLAG	true	Enables the DNS packet compression of each AXFR packet.
axfr-max-packet-size	INT	4096 bytes	The maximum size of an AXFR packet. (MIN: 512, MAX: 65535)
axfr-max-record-by-packet	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.
axfr-retry-delay	INT	600 sec	Number of seconds between each retry for the first transfer from the master name server.
axfr-retry-jitter	INT	180 sec	Jitter applied to axfr-retry-delay.
chroot	FLAG	off	Enabling this flag will make the server jail itself in the chroot-path directory.
chroot-path	PATH	/	The directory used for the jail.
cpu-count-override	INT	0	Overrides the detected number of logical cpus (0 : automatic, MAX: 256).
daemon	FLAG	true	Enabling this flag will make the server detach from the console and work in background.
data-path	PATH	/var/zones	The base path were lies the data (base 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) (0 : automatic, MAX: 128)
edns0-max-size	INT	4096 bytes	EDNS0 packets size.
gid	GID	0	The group ID that the server will use.
keys-path	PATH	/var/zones/keys	The base path of the DNSSEC keys.
listen	HOST(S)	0.0.0.0	The list of interfaces to listen to.
log-path	PATH	/var/log	The base path where the log files are written.
max-tcp-queries	INT	5	The maximum number of parallel TCP connections, allowed. (MIN: 0, MAX: 512)
pid-file	STR	yadifa.pid	The pid file name.
pid-path	PATH	/var/run	The path for the pid file.
queries-log-type	INT	1	Query log format. (0: none, 1: YADIFA format, 2: BIND format, 3: YADIFA and BIND format at once)
server-port,port	INT	53	The default dns port. (MIN: 1, MAX:65535)
sig-validity-interval	INT	31 days	The number of hours for which an automatic signature is valid. (MIN: 7 days , MAX: 366 days)
sig-validity-jitter, sig-jitter	INT	3600 sec	The signature expiration validity jitter in seconds (1 hour). (MIN: 0, MAX: 86400 sec)
sig-validity-regeneration	INT	auto hours	Signatures expiring in less than the indicated amount of hours will be recomputed. (MIN: 24 hours, MAX: 168 hours, default: chosen by YADIFA)
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, MAX: 31 days)
tcp-query-min-rate	INT	4096 bytes / sec	The minimum rate required in a TCP connection (read and write). Slower connections are closed. The units are bytes per second.

thread-count-by-address	INT	0	Number of independent threads used to process each listening address. (0: single threaded, MAX: number of CPU's, -1: YADIFA chooses)
uid	UID	0	The user ID that the server will use.
version-chaos	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.
xfr-path	PATH	/var/zones/xfr	The base path used for AXFR and journal storage.

Table 10.2: Parameters main section

For 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                yadifa.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

        server-port             53
        listen                  192.0.2.53, 192.0.2.153 port 8053

        uid                     yadifad
        gid                     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>
```

The ‘zone’ sections

Each zone is defined by one section only.

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

For example:

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

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

The ‘key’ sections

Each TSIG key must be defined by one section.

For example:

PARAMETER	TYPE	DEFAULT	DESCRIPTION
notify-auto	FLAG	TRUE	(TRUE: **DNS NOTIFY**[9] will be send to all name servers in APEX, FALSE: the content of APEX will be ignored)
no-master-updates	FLAG	FALSE	If set to true, the slave will not probe nor download changes from the master.
notify-retry-count	INT	5	Number of times YADIFA tries to send a **DNS NOTIFY**[9] .
notify-retry-period	INT	1	Time period between two **DNS NOTIFY**[9] attempts.
notify-retry-period-increase	INT	0	Increase of the time period between two **DNS NOTIFY**[9] attempts.
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.
allow-control	ACL	as main	Control commands control list. Only the matching sources are allowed.
also-notify	HOST(S)	-	The list of servers to notify in the event of a change. Currently only used by masters when a dynamic update occurs.
dnssec-mode	DNSSECTYPE	none	Type of DNSSEC used for the zone. As master name sever, YADIFA will try to maintain that state.
domain	FQDN	-	Mandatory. Sets the domain of the zone (i.e.: eurid.eu).
file-name, file	PATH	-	Sets the zone file name. Only mandatory for a master zone.
master	HOST	-	Mandatory for a slave. Sets the master server. Only one is supported.
sig-validity-interval	INTEGER	as main	The number of hours for which an automatic signature is valid. (MIN: 7 days , MAX: 366 days)
sig-validity-jitter, sig-jitter	INTEGER	as main	The signature expiration validity jitter in seconds (1 hour). (MIN: 0, MAX: 86400 sec)
sig-validity-regeneration	INTEGER	as main	The signatures expiring in less than the indicated amount of hours will be recomputed. (MIN: 24 hours, MAX: 168 hours, default: chosen by YADIFA)
type	ENUM	-	Mandatory. Sets the type of zone : either 'master' or 'slave'.

Table 10.3: Parameters zone sections

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.

Table 10.4: Parameters key sections

```
<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>
```

The '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:

```
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:

  ```
  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:

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

For 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>
```

The 'channels' section

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

- file

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

Table 10.5: Parameters syslog

- 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.

  Supported facilities:

  Supported options:

  Note:

  For more information: man syslog

  For example:

```
syslog  syslog  CRON,PID
```

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

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.

Table 10.6: Parameters for channels

For example:

```
default-output STDOUT
```

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

  For example:

```
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:

```
yadifa yadifa.log 0644
```

For 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>
```

The '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:

The current set of levels is:

Note:

> Messages at the 'crit', 'alert' and 'emerg' levels do trigger an automatic shutdown of the server.

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.)
statistics	Internal statistics periodic output
system	Low-level output (thread management, task scheduling, timed events)
zone	Internal zone loading output
queries	Queries output

Table 10.7: logger sources

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

Table 10.8: logger levels

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

For example:

```
<loggers>
        # info, notice and warning level messages from the database logging
        # will be outp
        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 'nsid' section

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

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

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

```
make
make install
```

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

Table 10.9: Parameters nsid section

For example:

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

For example:

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

The 'rrl' section

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

```
./configure --enable-rrl
```

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

```
make
make install
```

TYPE	DESCRIPTION

Table 10.10: Types

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

For 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>
```

11 Zones

Only textual zones are implemented.

The format of a zone file is defined in **RFC 1034**[6] and **RFC 1035**[7].

For example:

```
;; 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
```

11.1 MACROS

Some macros are implemented:

- @

- $TTL
- $ORIGIN

11.1.1 @

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:

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

For example:

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

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

11.1.2 $TTL

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

For example:

```
;; 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
```

11.1.3 $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.

For example:

```
;; 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
```

11.2 Classes

YADIFA knows only one class:

- IN [7].

11.3 Resource record types

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

- SOA
- A
- AAAA
- CNAME
- DNSKEY
- DS
- HINFO
- MX
- NAPTR
- NS
- NSEC
- NSEC3
- NSEC3PARAM
- PTR
- RRSIG
- SRV
- SSHFP
- TLSA
- TXT
- WKS.

12 Statistics

YADIFAD has a range of statistics available with one configuration setting. The statistics values are grouped into inputs, outputs and the RRL. Groups are 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:

```
udp (in=303 qr=303 ni=0 up=0 dr=0 st=91191 un=0) tcp (in=369 qr=368 ni=0 up=0 dr=0 st=82477 un=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 messages 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

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 messages 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

Bibliography

[1] R. Arends. *Resource Records for the DNS Security Extensions*, March 2005. **RFC 4034**.

[2] R. Arends. *NS Security (DNSSEC) Hashed Authenticated Denial of Existence*, March 2008. **RFC 5515**.

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

[4] S. Kwan. *Secret Key Transaction Authentication for DNS (GSS-TSIG)*, October 2003. **RFC 3645**.

[5] E. Lewis. *DNS Zone Transfer Protocol (AXFR)*, June 2010. **RFC 5936**.

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

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

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

[9] Paul Vixie. *DNS NOTIFY*, August 1996. **RFC 1996**.

[10] Paul Vixie. *DNS UPDATE*, April 1997. **RFC 2136**.

[11] Paul Vixie. *EXTENSION MECHANISMS FOR DNS (EDNS0)*, August 1999. **RFC 2671**.

Index