Unbound User Manual

NLnet Labs

May 31, 2021

Getting Started

Warning: This project has the goal to replace the existing documentation with an open source community project. **This is a work in progress.** A skeleton and table of contents is in place but not every page has been written.

Do you love to write and know your way around DNS and Unbound? Help us expand unbound.readthedocs.io and we'll compensate you for your time. Contact us at docs@nlnetlabs.nl to apply.

Welcome to the Unbound documentation. Unbound is a validating, recursive, caching DNS resolver. It is designed to be fast and lean and incorporates modern features based on open standards.

Unbound runs on FreeBSD, OpenBSD, NetBSD, MacOS, Linux and Microsoft Windows, with packages available for most platforms. It is included in the base-system of FreeBSD and OpenBSD and in the standard repositories of most Linux distributions. Installation and configuration is designed to be easy. Setting up a resolver for your machine or network can be done with only a few lines of configuration.

This documentation is an open source project maintained by NLnet Labs. is edited via text files in the reStructured-Text markup language and then compiled into a static website/offline document using the open source Sphinx and ReadTheDocs tools.

We always appreciate your feedback and improvements. You can submit an issue or pull request on the GitHub repository, or post a message on the Unbound users mailing list.

CHAPTER 1

Installation

1.1 Compiling

If your distribution package manager has a package for unbound you can skip this step, just install the package with your package manager.

To compile the software you need to have `openssl`, and its include files (from a package often called `openssl-devel`). Run `./configure [options]; make; make install`

If you do not have the `libldns` library installed, a version is included with the unbound source tarball, which is automatically used.

Options for configure. You can customize the default config locations for various files and directories, as well as the install location for the program with `--prefix=/usr/local`. You can specify `--with-ldns=dir` or `--with-libevent=dir` or `--with-ssl=dir` to link with the library at that location. Unless you want to tweak things, no options are needed for `./configure`.

On some BSD systems you have to use gmake instead of make.

You can install with `make install`, uninstall with `make uninstall`. The uninstall does not remove the config file.

In the contrib directory in the unbound source are sample rc.d scripts for unbound (for BSD and Linux type systems).

1.2 Setup

The config file is copied into `/usr/local/etc/unbound/unbound.conf` but some distributions may put it in `/etc/unbound/unbound.conf` or `/etc/unbound.conf`. The config file is fully annotated, you can go through it and select the options you like. Or you can use the below, a quick set of common options to serve the local subnet.

A common setup for DNS service for an IPv4 subnet and IPv6 localhost is below. You can change the IPv4 subnet to match the subnet that you use. And add your IPv6 subnet if you have one.

```
# unbound.conf for a local subnet.
server:
    interface: 0.0.0.0
    interface: ::0
    access-control: 192.168.0.0/16 allow
    access-control: ::1 allow
    verbosity: 1
```

By default the software comes with chroot enabled. This provides an extra layer of defence against remote exploits. Enter file paths as full pathnames starting at the root of the filesystem (`/`). If chroot gives you trouble, you can disable it with `chroot: ""` in the config.

Also the server assumes the username `unbound` to drop privileges. You can add this user with your favourite account management tool (`useradd(8)`), or disable the feature with `username: ""` in the config.

Start the server using the rc.d script (if you or the package manager installed one) as `/etc/rc.d/init.d/unbound start`. Or `unbound -c <config>` as root.

1.3 Setup Remote Control

If you want to you can setup remote control using `unbound-control`. First run `unbound-control-setup` to generate the necessary TLS key files (they are put in the default install directory). If you use a username of `unbound` to run the daemon from use `sudo -u unbound unbound-control-setup` to generate the keys, so that the server is allowed to read the keys. Then add the following at the end of the config file.

```
# enable remote-control
remote-control:
    control-enable: yes
```

You can now use `unbound-control` to send commands to the daemon. It needs to read the key files, so you may need to `sudo unbound-control`. Only connections from localhost are allowed by default.

CHAPTER 2

Configuration

Minimal “sane” config:

- ACL
- unbound-control-setup
- unbound-anchor

CHAPTER 3

Resolver for Enterprise Networks

Using Unbound as the resolver for the local network of businesses of all sizes.

CHAPTER 4

Resolver for Home Networks

To start off, let's ask the all-important question "Why would you want Unbound as a resolver for your home network?" Firstly, Unbound supports DNSSEC which, through an authentication chain, verifies that the DNS query responses you receive are unaltered, as opposed to query responses which are not DNSSEC-signed and could be changed by anyone who has access to the query. Secondly, by using your own resolver you stop sharing your DNS traffic with third parties (your ISP, Google, Cloudflare, Quad9, etc.) and increase your DNS privacy. While you still send out (parts of) your queries unencrypted, you could configure Unbound to take it a step further, which we'll talk about in an upcoming guide. Lastly, when you run your own resolver your DNS cache will be local to your network. Even though the first time you resolve a domain name may be slightly slower than using your ISP's resolver, all subsequent queries for the name will likely be much faster.

In this tutorial we'll look at setting up Unbound as a DNS resolver; Firstly for your own machine, and then for your entire network.

4.1 Setting up Unbound

Unbound is a powerful validating, recursive, caching DNS resolver. It's used by some of the biggest tech companies in the world as well as small-office / home-office users, who use it together with ad blockers and firewalls, or self-hosted resolvers. Setting it up for your home network can be quite simple as we'll showcase below.

Setting up a caching DNS server for your entire home network requires a recursive DNS resolver, and a dedicated machine on which the resolver runs; this (small) system is always on and accessible to the entire network. It can be as small as a Raspberry Pi or any other available Linux/Unix machine that is always online and has Internet connectivity via your router.

Because of the variety of machines that Unbound can run on we cannot create a comprehensive tutorial for all possible options. For this tutorial we will use **`Ubuntu 20.04.1 LTS`** as a stepping stone you can adapt and apply to other systems.

While you could download the code from GitHub and build it yourself, getting a copy can be as simple as running:

```
sudo apt update
sudo apt install unbound -y
```

This gives you a complete and running version of Unbound which behaves as a caching recursive DNS resolver out of the box for the system on which you install it. You can check which version of Unbound you have installed with `unbound -V`. The version installed will vary depending on the operating system. If the version is installed is quite old (at the time of writing it isn't) or you'd simply like to run the latest and greatest version you can download the latest release tarball from our website and build it yourself.

Do note that by default the DNS server will be queryable only from the local host, i.e. the system on which you installed Unbound. We will change that later.

4.2 Testing the resolver locally

To verify that the server works correctly, it's a good idea to test it before committing the entire network to it. Luckily we can test this on the machine that you installed Unbound on (locally) and from any other machine (remotely) that will be using the resolver after we expose Unbound to the network.

The command for testing locally on the Unbound machine is:

```
dig example.com @127.0.0.1
```

Here we tell the **dig** tool to look up the IP address for example.com, and to ask for this information from the server running at the IP address `127.0.0.1`, which is where our Unbound machine is running by default. We can verify that Unbound has indeed answered our query instead of the default resolver that is present on Ubuntu by default. In the output of every **dig** command there is an `ANSWER SECTION` which gives the response to the query. In the footer section of the output, the server which has answered the query under the `SERVER` entry. The entry will look like:

```
;; SERVER: 127.0.0.1#53(127.0.0.1)
```

In the next section we will be disabling the default Ubuntu resolver. To verify that we do it correctly it is useful to know the address of the default resolver as a baseline. For this baseline we also use a **dig** query, but this time without specifying an IP address (which causes dig to use the machine's default DNS resolver).

```
dig example.com
```

While the response should be the same, the `SERVER` entry in the response should look like:

```
;; SERVER: 127.0.0.53#53(127.0.0.53)
```

Note that the final IPv4 digit is 53 and not 1, as with our Unbound instance.

4.3 Setting up for a single machine

Now that we have tested our Unbound resolver, we can tell our machine to use it by default. The resolver your machine uses by default is defined in `/etc/systemd/resolved.conf` in the `DNS` entry (It uses `127.0.0.53`). While just changing this file will work as long as the machine doesn't reboot, we need to make sure that this change is persistent. To do that, we need to change the `DNS` entry to be equal to `127.0.0.1` so the machine uses Unbound as default. To make the change persistent, we also need to set the `DNSStubListener` to `no` so that is not changed by our router (such as with a "recommended resolver" mentioned below). We also want to enable the `DNSSEC` option so that we can verify the integrity the responses we get to our DNS queries. With your favourite text editor (e.g. **nano**) we can modify the file:

```
nano /etc/systemd/resolved.conf
```

Here, under there `[Resolve]` header we add (or rather enable by removing the "#") the options:

```
[Resolve]
DNS=127.0.0.1
#FallbackDNS=
#Domains=
DNSSEC=yes
#DNSOverTLS=no
#MulticastDNS=no
#LLMNR=no
#Cache=no-negative
DNSStubListener=no
#DNSStubListenerExtra=
```

With this file modified, we can start using this configuration with:

```
systemctl restart systemd-resolved
```

Now the operating system should use our Unbound instance as default. A quick test a **dig** without specifying the address of the Unbound server should give the same result as specifying it did above (with `@127.0.0.1`).

```
dig example.com
```

Note that the "SERVER" section in the output from **dig** should also contain the local IP address of our server.

```
;; SERVER: 127.0.0.1#53(127.0.0.1)
```

4.4 Setting up for the rest of the network

While we currently have a working instance of Unbound, we need it to be reachable from within our entire network. With that comes the headache of dealing with (local) IP addresses. It's likely that your home router distributed local IP addresses to your devices. If this is the case (i.e. you didn't change it by hand), they should be **RFC 1918** ranges:

```
10.0.0.0 - 10.255.255.255 (10/8)
172.16.0.0 - 172.31.255.255 (172.16/12)
192.168.0.0 - 192.168.255.255 (192.168/16)
```

To find the IP address of the machine that is running Unbound, we use:

```
hostname --all-ip-addresses
```

If you just have one IP address as output from the **hostname** command that will be the correct one. If you have multiple IP addresses, the easiest way to determine which IP address to use, is to find out which connection goes to your home router. Keep in mind that using the wrong IP address here can be a source of connectivity errors further on. For the purpose of this tutorial we assume that our home router has the IP address `192.168.0.1`, as this is typical for home routers, and our resolver machine (the machine that is running our Unbound instance) has IP address `192.168.0.2`, which we will get into in the next section.

As a prerequisite for the next step, we need to configure our Unbound instance to be reachable from devices other than only the machine on which the Unbound is running. Unbound is a highly capable resolver, and as such has many options which can be set; the full example config is almost 1200 lines long, but we'll need but a fraction of these settings. (If you are interested, all configurables are documented in the extensive manual page of *unbound.conf*).

The default config is found at:

```
/etc/unbound/unbound.conf
```

If you open this for the first time it looks very empty. It is still usable as a resolver for one machine, as this is how the Unbound defaults are configured. It's not, however, enough for our purposes, so we will add the minimal configuration options needed.

The options that we add to the current config file to make it a "minimal usable config" are as follows. Note that the IPv6 options are commented out, but we recommend to uncomment them if your router and network supports it.

```
server:
                # location of the trust anchor file that enables DNSSEC
                auto-trust-anchor-file: "/var/lib/unbound/root.key"
                # send minimal amount of information to upstream servers to enhance␣
↪privacy
                qname-minimisation: yes
                # the interface that is used to connect to the network (this will␣
↪listen to all interfaces)
                interface: 0.0.0.0
                # interface: ::0
                # addresses from the IP range that are allowed to connect to the␣
↪resolver
                access-control: 192.168.0.0/16 allow
                # access-control: 2001:DB8/64 allow

remote-control:
                # allows controling unbound using "unbound-control"
                control-enable: yes
```

The interface is currently configured to listen to any address on the machine, and the access-control only allows queries from the `192.168.0.0/16` IP subnet range. Note that the IP address we chose above (`192.168.0.1` and `192.168.0.2`) fall within the `192.168.0.0/16` range.

To prepare our config we are going to modify the existing config in `/etc/unbound/unbound.conf`. If you open the file for the first time, you see that there is already an "include" in there. The "include" enables us to do DNSSEC, which allows Unbound to verify the source of the answers that it receives, as well as QNAME minimisation. For convienience these configuration options have already been added in the minimal config. The config also includes the **`remote-control`** in the config to enable controlling Unbound using **`unbound-control`** command which is useful if you want to modify the config later on.

Using the text editor again, we can then add the minimal config shown above, making any changes to the access control where needed.When we've modified the configuration we check it for mistakes with the **`unbound-checkconf`** command:

```
unbound-checkconf unbound.conf
```

If this command reports no errors, we need to stop the currently running Unbound instance and restart it with our new configuration. You can stop Unbound with:

```
sudo pkill -f unbound
```

And you can restart Unbound with:

```
unbound-control start
```

From this point on, we can **`stop`**, **`start`**, and **`reload`** Unbound with **`unbound-control`** if you want to make changes to the configuration.

4.5 Testing the resolver from a remote machine

So now we have a DNS resolver which should be reachable from within the network. To be able to verify that our resolver is working correctly, we want to test it from anoither machine in the network. As mentioned above, this tutorial uses the address `192.168.0.2` (not `127.0.0.1` as we saw earlier) as an example for the machine running Unbound. Armed with the IP address we can send a query to our DNS resolver from another machine which is within our home network. To do this we use the same dig command, only we change the IP address where the query is asked.

```
dig example.com @192.168.0.2
```

This should give the same result as above. The `SERVER` entry in the footer reflects from which server the response was received.

4.6 Where it all comes together

We should now have a functioning DNS resolver that is accessible to all machines in our network (**make sure you do before you continue**).

The next step then is a little tricky as there are many options and variations possible. We have a choice of which machines in our network will be using our configured DNS resolver. This can range from a single machine to all the machines that are connected. Since this tutorial cannot (and does not try to) be comprehensive for the range of choices, we will look at some of the basic examples which you can implement and expand on.

Most machines when they first connect to a network get a "recommended resolver" from your router using DHCP (Dynamic Host Configuration Protocol). To change this, we need to log into the router. Earlier in this tutorial we assume the home router was using `192.168.0.1`, though in reality this can differ. if this does differ, the unbound config needs to be changed as well. To find the IP address of our home router, which is likely be under the `default gateway` entry from:

```
ip route
```

When you've found the IP address of your home router, you can copy the address to a web browser, which should give you access to the router configuration portal. If you can't find the portal using this method, consult the manual or the manufacturer's website. When you have access, you should change the DHCP configuration to advertise the IP address of the machine running Unbound as the default gateway. In the case of our example, that would be `192.168.0.2`.

Another possibility is a machine that does not use a resolver that is "recommended" by your router. This machine can be running its own resolver or be connected to a different one altogether. If you want these machines to use the Unbound resolver you set up, you need to change the configuration of the machine.

CHAPTER 5

Resolver for ISP Networks

Using Unbound as the resolver in a network of an ISP.

CHAPTER 6

Stub Resolver of a Single Machine

(Is this even a stub resolver or rather a local forwarder?)

CHAPTER 7

Maximum Privacy Resolver

Make Unbound as privacy-friendly as possible.

CHAPTER 8

Privacy

Unbound offers various ways to enhance your privacy. This section covers multiple ways of achieving this.

8.1 Encryption

To do.

8.2 QNAME Minimisation

To do.

8.3 Aggressive NSEC

Unbound has implemented the aggressive use of the DNSSEC-Validated cache, also known as *Aggressive NSEC*, based on **RFC 8198**. This section first describes how NSEC works, and then covers how synthesised answers can be generated based on the DNSSEC-Validated cache.

8.3.1 Introduction

DNS relies heavily on caching. A lot of performance can be gained by storing answers to previous queries close to the client. If an authoritative name server would have to be queried for every single request, performance would be severely impacted.

In addition to caching the positive answer to queries, negative answers are also cached. These negative answers are an acknowledgement from the name server that a name does not exist (an answer with the response code set to NXDOMAIN) or that the type in the query does not exist for the name in the query. The latter is known as an answer with the NODATA pseudo response code, as specified in section 1 of **RFC 2308**.

8.3.2 NSEC (Next Secure) Records

DNSSEC is not only used to prove the authenticity of records in a DNS answer by verifying the DNSSEC signatures of the records, it is also used to prove the absence of records. DNSSEC uses NSEC (next secure), as well as NSEC3 records for these proof of non-existence answers. An NSEC record indicates that there are no records that are sorted between the two domain names it contains. The canonical DNS name order is used for the sorting, as described in section 6.1 of **RFC 4034**. An NSEC record also has a type bitmap which specifies the record types that exist for the owner name of the NSEC record. Like any other DNS record, the authenticity of NSEC records can be validated using its DNSSEC signature which is located in the RRSIG record.

This NSEC record is taken as an example:

```
golf.example.net NSEC kilo.example.net TXT RRSIG NSEC
```

The record indicates that the owner name `golf.example.net` exists and that the owner name has records for the *TXT*, *RRSIG* and *NSEC* types. It therefore proves that there is no *A* record for `golf.example.net`. This NSEC record also proves that there are no records alphabetically sorted between its owner and its next domain name (kilo.example.net). This record therefore proves that there is no record for juliett.example.net.

8.3.3 DNSSEC Signatures on Wildcard Records

Wildcard expansion on NSEC record is specifically allowed by **RFC 4592**. In order to answer a DNS query using a wildcard record, an authoritative nameserver replaces the owner name of the wildcard record with the name in the query. DNSSEC is designed in such way that it can sign a complete zone before it starts serving. Because the query name that will be used for the wildcard record is not known when the zone is being signed, it is not possible to make a DNSSEC signature for it. Therefore the original owner name with the wildcard label is used for the signature.

The labels field that is part of the RRSIG record indicates the number of labels of the owner name without the wildcard label. This labels field can be used by a DNSSEC validator to detect that this is a signature for a wildcard record. A DNSSEC validator then knows it needs to validate the signature using the original wildcard owner, and not the expanded owner that matches the query name. A validator gets the original owner name by taking the number of rightmost labels defined in the labels field from the expanded owner and then prepend it with the wildcard label `*`.

This is an example of a wildcard expanded RRSIG record, with the signature omitted to keep the text compact:

```
zebra.example.net.  2710 IN RRSIG NSEC 8 2 10200 20180226143900 20180129143900 42393␣
↪example.net. [..]
```

This RRSIG record has a label count of two, while the number of labels in the owner name (excluding the root label) is three. Using that information a validator will take the two last labels of the owner name (example.net) and prepend the wildcard label to it. It therefore checks the signature using the original wildcard name, which is `*.example.net`.

8.3.4 Generating NODATA Answers

The traditional Unbound cache implementation is based on exactly matching cached messages to the query name, query type and query class. If a client asks for a *TXT* record for `example.net`, the resolver will search the cache and if that fails go and look up the answer at the authoritative name server. This query to the authoritative name server will result in a response containing the existing *TXT* record. If the resolver now receives a query for the same name but for the *TLSA* type, the resolver will check its cache, in this case can not find a matching record in the cache and will, as a result, send a query to the authoritative name server. That name server will now reply with a NODATA answer, indicating that the `example.net` name does exist, but there is no record for that name with the TLSA record. A third query for the same name for another non-existing type, for example *SRV*, will once again not result in a cache hit and will generate yet another query with again a NODATA answer as result.

In this example the `example.net` zone is DNSSEC signed. This means the absence of these records need to be proven using NSEC records. NSEC records indicate which types exist for a name and which names exist in a zone. NSEC records have a cryptographic signature which make them tamper proof. By knowing the existing record and types in a zone, a DNSSEC validator can prove that the combination of query name and query type indeed does not exist.

The NODATA answer for the `example.net` name with the *TLSA* query type could for example contain this NSEC record:

```
example.net. 3600 IN NSEC !.example.net. A NS SOA MX TXT AAAA NAPTR RRSIG NSEC DNSKEY
```

This record proves which types exist for `example.net` (*A, NS, SOA* etc.) and thereby proves that the *TLSA* record indeed does not exist. The NODATA response to the third query in above example (the *SRV* query for `example.net`) will contain exactly the same NSEC record to prove the absence of the *SRV* record. Because this NSEC record was already cached after the lookup for the TLSA record we could have used that already obtained NSEC record to generate a DNSSEC secure answer, without the need to send another query to the authoritative name server.

Important: To use previously cached NSEC records to generate responses in Unbound, use the *aggressive-nsec* option in unbound.conf:

```
aggressive-nsec: yes
```

8.3.5 Generating NXDOMAIN Answers

An answer with the NXDOMAIN response code indicates that a name does not exist at all, which is also proven using an NSEC record. If `example.net` would contain these alphabetically sorted records (some simplification ahead):

```
example.net.            IN SOA [..]
                        IN NS alfa.example.net.
alfa.example.net.       IN A 198.51.100.52
sierra.example.net.     IN A 198.51.100.98
```

then DNSSEC would make sure these NSEC records are inserted and signed:

```
example.net.          IN NSEC alfa.example.net.   NS SOA DNSKEY
alfa.example.net.     IN NSEC sierra.example.net. A
sierra.example.net.   IN NSEC example.net.        A
```

They attest that no name exists between `alfa.example.net` and `sierra.example.net`. So if you query for `lima.example.net`, you will get back the NXDOMAIN from the authoritative name server, as well as the NSEC record for `alfa.example.net` — `sierra.example.net` as proof that the query name does not exist and the NSEC record for `example.net` — `alfa.example.net` as proof that the `*.example.net` wildcard record does not exist.

If the user now queries for for `delta.example.net`, resolvers would normally ask the authoritative server again because there is no message cached for that name. But because the NSEC records for `alfa.example.net` — `sierra.example.net` and `example.net` — `alfa.example.net` are already cached, the implementation of **RFC 8198** will allow Unbound to deduce that it doesn't need to send a new query. It is already able to prove that the name doesn't exist and immediately, or *aggressively* if you will, returns an NXDOMAIN answer.

8.3.6 Generating Wildcard Answers

There is one more type of message that can be generated using cached NSEC records, namely wildcard answers. A DNSSEC validator only accepts a wildcard answer when there is proof that there is no record for the query name. When we have this zone containing a wildcard record:

```
example.net.           IN SOA [..]
                       IN NS alfa.example.net.
*.example.net.         IN TXT "A wildcard record"
alfa.example.net.      IN A 198.51.100.52
sierra.example.net.    IN A 198.51.100.98
```

then a TXT query for `delta.example.net` will be answered with the following records, indicating that there is no direct match for the query name but that there is a matching wildcard record:

```
;; ANSWER SECTION:
delta.example.net.     IN TXT "A wildcard record"
delta.example.net.     IN RRSIG TXT 8 2 [..]

;; AUTHORITY SECTION:

alfa.example.net.      IN NSEC sierra.example.net.   A
```

The `alfa.example.net — sierra.example.net` NSEC record indicates that there is no `delta.example.net` record. The labels field in the signature indicates that the returned TXT record is expanded using the `*.example.net` record.

Unbound uses this knowledge to store the wildcard RRset also under the original owner name, containing the wildcard record, when aggressive use of NSEC is enabled. After receiving a query for `echo.example.net` Unbound finds the NSEC record proving the absence in its cache. Unbound will then look in the cache for a `*.example.net` *TXT* record, which also exists. These records are then used to generate an answer without sending a query to the name server.

Note: Aggressive NSEC can result in a reduction of traffic on all levels of the DNS hierarchy but it will be most noticeable at the root, as typically more than half of all responses are NXDOMAIN.

Another benefit of a wide deployment of aggressive NSEC is the incentive to DNSSEC sign your zone. If you don't want to have a large amount of queries for non-existing records at your name server, signing your zone will prevent this.

8.4 Auth Zone

To do.

CHAPTER 9

Performance

CHAPTER 10

Filtering and Manipulating Data

Unbound offers various ways to filter queries and manipulate data. This section covers various methods to achieve this, with practical examples.

10.1 Local Zones and Local Data

To do.

10.2 Tags and Views

The tags and views functionality make it possible to send specific DNS answers based on the IP address of the client.

10.2.1 Tags

The tags functionality makes it possible to divide client source addresses in categories (tags), and use local-zone and local-data information for these specific tags.

Before these tags can be used, you need to define them in the Unbound configuration using *define-tags*. In this example, a tag for domains containing malware is set, along with one for domains of gambling sites:

```
define-tags: "malware gambling"
```

Now that Unbound is aware of the existing tags, it is possible to start using them. The *access-control-tag* element is used to specify the tag to use for a client addresses. You can add multiple tags to an access-control element:

```
access-control-tag: 10.0.1.0/24 "malware"
access-control-tag: 10.0.2.0/24 "malware"
access-control-tag: 10.0.3.0/24 "gambling"
access-control-tag: 10.0.4.0/24 "malware gambling"
```

Unbound will create an *access-control-tag* element with the "allow" type if the IP address block in the *access-control-tag* element does not match an existing *access-control*.

When a query comes in from an address with a tag, Unbound starts searching its local-zone tree for the best match. The best match is the most specific local-zone with a matching tag, or without any tag. That means that local-zones without any tag will be used for all clients and tagged local-zones only for clients with matching tags.

Adding tags to local-zones can be done using the *local-zone-tag* element:

```
local-zone: malwarehere.example refuse
local-zone: somegamblingsite.example static
local-zone: matchestwotags.example transparent
local-zone: notags.example inform

local-zone-tag: malwarehere.example malware
local-zone-tag: somegamblingsite.example malware
local-zone-tag: matchestwotags.example "malware gambling"
```

A local-zone can have multiple tags, as illustrated in the example above. The tagged local-zones will be used if one or more tags match the client. So, the matchestwotags.example local-zone will be used for all clients with at least the malware or gambling tag. The used local-zone type will be the type specified in the matching local-zone. It is possible to depend the local-zone type on the client address and tag combination. Setting tag specific local-zone types can be done using *access-control-tag-action*:

```
access-control-tag-action: 10.0.1.0/24 "malware" refuse
access-control-tag-action: 10.0.2.0/24 "malware" deny
```

In addition to configuring a local-zone type for some specific client address/tag match, it is also possible to set the used local-data RRs. This can be done using the *access-control-tag-data* element:

```
access-control-tag-data: 10.0.4.0/24 "gambling" "A 127.0.0.1"
```

Sometimes you might want to override a local-zone type for a specific IP address block, regardless the type configured for tagged and untagged local zones, and regardless the type configured using access-control-tag action. This override can be done using *local-zone-override*.

10.2.2 Views

Tags make is possible to divide a large number of local-zones in categories, and assign these categories to a large number of IP address blocks. As tags on the IP address blocks and local-zones are stored in bitmaps, it is advised to keep the number of tags low. If a lot of clients have their own local-zones, without sharing these to other IP address blocks, it can result a large amount tags. In this situation is is more convenient to give the client's IP address block its own tree containing local-zones. Another benefit of having a separate local zone tree is that it makes it possible to apply a local-zone action to a part of the domain space, without having other local-zone elements of subdomains overriding this. Configuring a client specific local-zone tree can be done using views.

A view is a named list of configuration options. The supported view configuration options are local-zone and local-data.

A view is configured using a view clause. There may be multiple view clauses, each with a unique name. For example:

```
view:
    name: "firstview"
    local-zone: example.com inform
    local-data: 'example.com TXT "this is an example"'
    local-zone: refused.example.nl refuse
```

Mapping a view to a client can be done using the *access-control-view* element:

```
access-control-view: 10.0.5.0/24 firstview
```

By default, view configuration options override the global configuration outside the view. When a client matches a view it will only use the view's local-zone tree. This behaviour can be changed by setting *view-first* to yes. If view-first is enabled, Unbound will try to use the view's local-zone tree, and if there is no match it will search the global tree.

10.3 Response Policy Zones

Response Policy Zones (RPZ) is a mechanism that makes it possible to define your local policies in a standardised way and load your policies from external sources.

10.3.1 Introduction

Unbound has support for local-zones and local-data. This makes it possible to give a custom answer back for certain domain names. It also contains the `respip` module which makes it possible to rewrite answers containing certain IP addresses. Although these options are heavily used, they are Unbound specific. If you operate resolvers from multiple vendors you have to maintain your policies for multiple configurations, which all will have their own syntax. Using the Unbound specific configuration also makes it challenging to consume policies from external sources. You will have to fetch the external policies in the offered format, and reformat it in such a way that Unbound will understand it. You then have to keep this list up-to-date, for example using unbound-control.

There is, however, a policy format that will work on different resolver implementations, and that has capabilities to be directly transferred and loaded from external sources: Response Policy Zones (RPZ).

10.3.2 RPZ Policies

RPZ policies are formatted in DNS zone files. This makes it possible to easily consume and keep them to up-to-date by using DNS zone transfers. Something that Unbound is already capable of doing for its auth-zone feature.

Each policy in the policy zone consists of a trigger and an action. The trigger describes when the policy should be applied. The action describes what action should be taken if the policy needs to be applied. Each trigger and action combination is defined as a Resource Record (RR) in the policy zone. The owner of the RR states the trigger, the type and RDATA state the action.

The latest RPZ draft describes five different policy triggers of which Unbound supports two: the QNAME trigger and the Response IP Address trigger.

10.3.3 QNAME Trigger

A policy with the *QNAME* trigger will be applied when the target domain name in the query (the query name, or QNAME) matches the trigger name. The trigger name is the part of the *owner* of the record before the origin of the zone. For example, if there is this record in the `rpz.nlnetlabs.nl` zone:

```
$ORIGIN rpz.nlnetlabs.nl.
example.com.rpz.nlnetlabs.nl.    TXT  "trigger for example.com"
```

then Unbound will add a policy for queries for `example.com`. Only exact matches for `example.com` will be triggered. If a policy for `example.com` is desired that includes all of its subdomains, this is possible by adding a wildcard record:

```
$ORIGIN rpz.nlnetlabs.nl.
example.com.rpz.nlnetlabs.nl.    TXT  "trigger for example.com"
*.example.com.rpz.nlnetlabs.nl.  TXT  "trigger for *.example.com"
```

10.3.4 RPZ Actions

The action that will be applied for above example is the *Local Data* action. This means that queries for `example.com` for the *TXT* type will be answered with the newly created record. Queries for types that do not exist in the policy zones will result in a NODATA answer.

```
$ drill txt example.com
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 14642
;; flags: qr aa rd ra ; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; example.com. IN TXT

;; ANSWER SECTION:
example.com. 3600 IN TXT "trigger for example.com"

$ drill aaaa example.com @127.0.0.54
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 4713
;; flags: qr aa rd ra ; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; example.com. IN AAAA

;; ANSWER SECTION:
```

Other RPZ actions that are supported by Unbound are the *NXDOMAIN*, *NODATA*, *PASSTHRU*, and *DROP* actions. All of these actions are defined by having a CNAME to a specific name. A policy for the NXDOMAIN action is created by having a CNAME to the root:

```
$ORIGIN rpz.nlnetlabs.nl.
example.com.rpz.nlnetlabs.nl.    CNAME .
```

The NXDOMAIN action will, as the name suggest, answer with an NXDOMAIN when triggered:

```
$ drill aaaa example.com
;; ->>HEADER<<- opcode: QUERY, rcode: NXDOMAIN, id: 14754
;; flags: qr aa rd ra ; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; example.com. IN AAAA
;; ANSWER SECTION:
```

The CNAME targets for the other RPZ actions are:

Action	RR type and RDATA
NXDOMAIN	CNAME .
NODATA	CNAME *.
PASSTHRU	CNAME rpz-passthru.
DROP	CNAME rpz-drop.

The NODATA action is self-explanatory. The DROP action will simply ignore (drop) the query. The PASSTHRU action makes it possible to exclude a domain, or IP address, from your policies. If the PASSTHRU action is triggered no other policy from any of the available policy zones will be applied:

```
$ORIGIN rpz.nlnetlabs.nl.
*.example.com.rpz.nlnetlabs.nl.   TXT "local data policy"
www.example.com.rpz.nlnetlabs.nl. CNAME rpz-passthru.
```

Queries for all subdomains of `example.com` will now be answered with an NXDOMAIN, except for queries for `www.example.com`, these will be resolved normally.

```
$ drill txt withpolicy.example.com
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 62993
;; flags: qr aa rd ra ; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; withpolicy.example.com. IN TXT

;; ANSWER SECTION:
withpolicy.example.com. 3600 IN TXT "local data policy"

$ drill txt www.example.com
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 42053
;; flags: qr rd ra ; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; www.example.com. IN TXT

;; ANSWER SECTION:
www.example.com. 86400 IN TXT "v=spf1 -all"
```

10.3.5 Response IP Address Trigger

The other RPZ trigger supported by Unbound is the *Response IP Address* trigger. This trigger makes it possible to apply the same RPZ actions as mentioned above, but triggered based on the IPv4 or IPv6 address in the answer section of the answer. The IP address to trigger on is again part of the owner of the policy records. The IP address is encoded in reverse form and prepended with the prefix length to use. This all is prepended to the `rpz-ip` label, which will be placed right under the apex of the zone. So, a trigger for addresses in the 192.0.2.0/24 block will be encoded as:

```
$ORIGIN rpz.nlnetlabs.nl.
24.0.2.0.192.rpz-ip.rpz.nlnetlabs.nl. [...]
```

IPv6 addresses can also be used in RPZ policies. In that case the `zz` label can be used to replace the longest set of zeros. A trigger for addresses in the 2001:DB8::/32 block will be encoded as:

```
$ORIGIN rpz.nlnetlabs.nl.
32.zz.db8.2001.rpz-ip.rpz.nlnetlabs.nl. [...]
```

It is possible to replace an address by applying one specified in a policy containing a Local Data action. For example, the IPv4 address for `example.com` is currently `93.184.216.34`, and can be changed to `192.0.2.1` like this:

```
$ORIGIN rpz.nlnetlabs.nl.
32.34.216.184.93.rpz-ip.rpz.nlnetlabs.nl. A 192.0.2.1

---

$ drill example.com
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 13670
;; flags: qr rd ra ; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; example.com. IN A
```

(continues on next page)

(continued from previous page)

```
;; ANSWER SECTION:

example.com. 3600 IN A 192.0.2.1
```

10.3.6 Implementation

The RPZ implementation in Unbound depends on the `respip` module, this module needs to be loaded using `module-config`. Each policy zone is configured in Unbound using the `rpz` clause. A minimal configuration with a single policy zone can look like:

```
server:
  module-config: "respip validator iterator"
rpz:
  name: rpz.nlnetlabs.nl
  zonefile: rpz.nlnetlabs.nl
```

In above example the policy zone will be loaded from file. It is also possible to load the zone using DNS zone transfers. Both AXFR and IXFR is supported, all additions and deletion in the zone will be picked up by Unbound and reflected in the local policies. Transferring the policy using a DNS zone transfer is as easy as specifying the server to get the zone from:

```
server:
  module-config: "respip validator iterator"
rpz:
  name: rpz.nlnetlabs.nl
  master: <ip address of server to transfer from>
  zonefile: rpz.nlnetlabs.nl
```

The zone will now be transferred from the configured address and saved to a zonefile on disk. It is possible to have more than one policy zone in Unbound. Having multiple policy zones is as simple as having multiple `rpz` clauses:

```
server:
  module-config: "respip validator iterator"
rpz:
  name: rpz.nlnetlabs.nl
  zonefile: rpz.nlnetlabs.nl
rpz:
  name: rpz2.nlnetlabs.nl
  zonefile: rpz2.nlnetlabs.nl
```

The policy zones will be applied in the configured order. In the example, Unbound will only look at the `rpz2.nlnetlabs.nl` policies if there is no match in the `rpz.nlnetlabs.nl` zone. If there is no match in any of the configured zones Unbound will continue to resolve the domain by sending upstream queries. Note that a PASSTHRU action is considered a match, having that action in the first zone will therefore stop Unbound from looking further at other policy zones.

Unbound has the possibility to override the actions that will be used for policies in a zone that matches the zone's triggers. This can be done using the `rpz-action-override` configuration option. The possible values for the option are: `nxdomain`, `nodata`, `passthru`, `drop`, `disabled`, and `cname`. The first four options of this list will do the same as the RPZ actions with the same name.

The `cname` override option will make it possible to apply a local data action using a CNAME for all matching triggers in the policy zone. The CNAME to use in the answer can be configured using the `rpz-cname-override` configuration option. Using these overrides are nice if you use an external feed to get a list of triggers, but would like to redirect all your users to your own domain:

```
RPZ zone (rpz.nlnetlabs.nl):
$ORIGIN rpz.nlnetlabs.nl.
drop.example.com.rpz.nlnetlabs.nl. CNAME rpz-drop.
32.34.216.184.93.rpz-ip.rpz.nlnetlabs.nl. A 192.0.2.1

---

Unbound config:
server:
  module-config: "respip validator iterator"

rpz:
  name: rpz.nlnetlabs.nl
  zonefile: rpz.nlnetlabs.nl
  rpz-action-override: cname
  rpz-cname-override: "example.nl."

---

Example queries:
$ drill drop.example.com
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 14547
;; flags: qr aa rd ra ; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; drop.example.com. IN A

;; ANSWER SECTION:
drop.example.com. 3600 IN CNAME example.nl.
example.nl. 3600 IN A 94.198.159.35

$ drill example.com
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 31187
;; flags: qr rd ra ; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; example.com. IN A

;; ANSWER SECTION:
example.com. 3600 IN CNAME example.nl.
example.nl. 3568 IN A 94.198.159.35
```

The `disabled` option will stop Unbound from applying any of the actions in the zone. This, combined with the `rpz-log` option, is a nice way to test what would happen to your traffic when a policy will be enabled, without directly impacting your users. The difference between `disabled` and `passthru` is that disabled is not considered to be a valid match and will therefore not stop Unbound from looking at the next configured policy zone.

When `rpz-log` is set to yes, Unbound will log all applied actions for a policy zone. With `rpz-log` enabled you can specify a name for the log using `rpz-log-name`, this way you can easily find all matches for a specific zone. It is also possible to get statistics per applied RPZ action using `unbound-control stats`. This requires the `extended-statistics` to be enabled.

Unbound's RPZ implementation works together with the tags functionality. This makes is possible to enable (some of) the policy zones only for a set of the users. To do this the tags need to be defined using `define-tag`, the correct tags need to be matched with the client IP addresses using `access-control-tag`, and the tags need to be specified for the policy zones for which they apply.

```
server:
  module-config: "respip validator iterator"
```

(continues on next page)

(continued from previous page)

```
  define-tag: "malware social"
  access-control-tag 127.0.0.10/32 "social"
  access-control-tag 127.0.0.20/32 "social malware"
  access-control-tag 127.0.0.30/32 "malware"
rpz:
  name: malware.rpz.example.com
  zonefile: malware.rpz.example.com
  tags: "malware"
rpz:
  name: social.rpz.example.com
  zonefile: social.rpz.example.com
  tags: "social"
```

Queries from 127.0.0.1 will not be filtered. For queries coming from 127.0.0.10 only the policies from the social.rpz.example.com zone will be used, for 127.0.0.30 only the policies from the malware.rpz.example.com zone will be used, and queries originated from 127.0.0.20 will be subjected to the policies from both zones.

CHAPTER 11

Monitoring and Reporting

11.1 Statistics via *unbound-control*

11.2 DNS TAP

CHAPTER 12

EDNS Client Subnet

CHAPTER 13

Resiliency

AKA: How to deal with attacks etc.

13.1 Rate Limiting

13.2 ACLs

CHAPTER 14

Server Selection

CHAPTER 15

Standards Compliance

This section lists the RFCs and other standards that Unbound implements and where, how, and why it deviates from the standards.

CHAPTER 16

Architecture

How is Unbound structured, how does it process queries, etc.

CHAPTER 17

The Structure of the Source Code

CHAPTER 18

Python Modules

This section talks about Python modules for Unbound. Using Unbound from Python is discussed in the libunbound section.

CHAPTER 19

DNSSEC Trust Anchor Management

This will talk about unbound-anchor and 5011.

CHAPTER 20

Embedding Unbound: libunbound

Information about using libunbound both from C and Python.

CHAPTER 21

Reference

This section will basically contain what is in the manpage currently. We need to figure out a smart way to either generate this from the manpage or have a common source from which we generate both this and the manpage version.

CHAPTER 22

Indices and tables

- genindex
- search

Index