Unbound DNS Tutorial

A validating, recursive, and caching DNS server

A Quick Overview of Unbound: A DNS Server For The Paranoid

Unbound is a very secure validating, recursive, and caching DNS server primarily developed by NLnet Labs, VeriSign Inc, Nominet, and Kirei. The software is distributed free of charge under the BSD license. The binaries are written with a high security focus, tight C code, and a mind set that it is always under attack, or remote servers are always trying to pass it bad information.

Unbound's design is a set of modular components which incorporate features including enhanced security (DNSSEC) validation, Internet Protocol Version 6 (IPv6), and a client resolver library API as an integral part of the architecture. Originally written for Posix-compatible Unix-like operating system, Unbound currently runs on FreeBSD, OpenBSD, NetBSD, and Linux, as well as Microsoft Windows.

The install and configuration of Unbound is incredibly easy. Most modern OS's already have Unbound packages made. We have verified that OpenBSD, FreeBSD, CentOS and Ubuntu have packages available through their current distribution methods. As for the configuration, a simple resolving caching DNS server which can be used for a single machine or multi-machine LAN is only a few lines long. Note that Unbound is not a full fledged authoritative server, but you can put in A records for forward and reverse resolution of a small private LAN.

In the future as BIND 10 is released it is expected that most open source operating systems like OpenBSD and Ubuntu will migrate over to Unbound as their primary DNS resolvers. BIND, also known as named, is getting extremely code bloated, slow and over complicated. The other problem is that BIND is used for 70% of DNS servers leading to a monoculture environment. If an attack or exploit comes out it is advantageous as the attacker to go after the most used software. Unbound is an incredibly fast and secure DNS name server which, due to its small size, can easily be code audited for security.

Lets take a look at some definitions and then some examples.

Common DNS functions

Before we examine the configuration examples, we need to understand the basic functions available through a modern DNS server. Then you can decide what type of DNS server you want and go right to the configurations below. Note that you can combine multiple functions together in a single DNS server. For example you can have a caching DNS, a recursive caching DNS, a validating recursive caching DNS, an authoritative validating recursive caching DNS, etc.

recursive DNS server

Caching name servers store DNS query results for a period of time determined in the configuration (time-to-live) of the domain name record in question. Recursive name servers resolve any query they receive, even if they are not authoritative for the question being asked, by consulting the server or servers that are authoritative for the query. Caching name servers, as seen in the next section, improve the efficiency of the DNS by reducing DNS traffic across the Internet, and by reducing load on authoritative name servers, particularly root name servers. Because they can answer questions more quickly, they also increase the performance of end-user applications that use the DNS.

A recursive DNS server will, on behalf of the client (resolver), traverse the paths of DNS across the Internet to retrieve the answer to the question. A simple query such as "What is the IP address of DNSwatch.COM ?" to a DNS server which supports recursive queries but is not authoritative for dnswatch.com would look something like the following:

- Your client resolver sends a query, "What is the IP address of dnswatch.com ?" to a locally configured DNS server like Unbound.

- Unbound DNS server looks up dnswatch.com in local tables (its cache) — not found if we have never asked for this hostname before.
- Unbound DNS sends a query to one of the root-servers in its root.hints file.
- The root-server replies with a referral to the TLD servers for ".com".
- Unbound sends a query, "What is the IP address dnswatch.com ?" to one of the .com TLD servers.
- The TLD server replies with a referral to the authoritative name servers for dnswatch.com at your Gateway, or Upstream providers Nameserver .
- Unbound sends query, "What is the IP address dnswatch.com ?" to an authoritative name server for dnswatch.com .
- The authoritative Zone file at your Gateway, or Upstream Providers Nameserver defines an "A" record which contains the IP address of dnswatch.com . Your Gateway, or Upstream Providers Nameserver returns the IP of dnswatch.com .
- Unbound receives the IP address of dnswatch.com , and returns the answer to the client resolver.
- Transaction complete.

As you can see a standard query for dnswatch.com is quite a bit of work and takes a little time to complete. That is why we like to keep a local copy of the answer on our local Unbound DNS server. That local copy is called a "cached" copy which leads to our next section.

caching DNS server

Caching name servers, also called DNS caches, are often also resolving name servers as they perform every step necessary to answer any DNS query they receive. To do this the name server queries each authoritative name server in turn, starting from the DNS root zone. The query process continues until Unbound reaches the authoritative server for the zone that contains the queried domain name. That server provides the answer to the question, or definitively says it can't be answered, and the caching resolver then returns this response to the client that asked the question.

This combination of resolver and cache creates a DNS server that will respond to look up requests by delivering answers from its cache if the hostname has been asked for before, or recursively resolving the hostnames if we have never seen this hostname. Cached results are returned in one (1), or two (2) milliseconds. While recursive queries can take hundreds of milliseconds or more.

validating DNS server

A validating DNS server is simply a resolver which verifies the response it has received is as correct as it can be sure of. This is usually accomplished using Secure DNS (DNSSEC) or using `0x20`-encoded random bits in the query to foil spoof attempts. Validation can also encompass sanity checks of the returned data, or by making sure a remote host does not try to return an illegal IP for an external hostname (dnsspoof).

To perform a cache poisoning attack for example, the attacker exploits a flaw in the DNS software. If the server does not correctly validate DNS responses to ensure that they are from an authoritative source (for example by using DNSSEC) the server will end up caching the incorrect entries locally and serve them to other users that make the same request. This just means that when you type in the hostname of your bank, you want to make sure that hostname matches you bank's actual ip address and not some phishing site in Cape Town, South Africa.

Paranoia in DNS security is also a reason you really need to have trust in any resolving caching server you do not control. We like to always have a DNS server under our control query the root servers and work their way down to the domain's authoritative server. This way we are sure of the setup as we are the administrators and we are confident in the DNS servers cached data due to the security design we have implemented. If you setup your DNS resolver to query the local ISP's dns cache you really, really need to trust their people and their setup is secure. From the view of an attacker, an ISP's dns cache is the perfect target. The attacker only needs to poison one dns server and all queries from all the users of that ISP go to the attackers compromised machines. This scenario has already happened once to AT&T's DNS cache servers (google for "dns attacks wild at&t") and it will sadly happen again.

authoritative DNS server

An authoritative DNS server is simply the primary owner of the hostname.

When a domain is registered with a domain name registrar, the zone administrator provides a list of name servers (typically at least two, for redundancy) that are authoritative for the zone that contains the domain. The registrar provides the names of these servers to the domain registry for the top level domain containing the zone. The domain registry in turn configures the authoritative name servers for that top level domain with delegations for each server for the zone. If the fully-

qualified domain name of any name server for a zone appears within that zone, the zone administrator provides IP addresses for that name server, which are installed in the parent zone as glue records; otherwise, the delegation consists of the list of NS records for that zone.

A name server server indicates that its response is authoritative by setting the Authoritative Answer (AA) bit in the response to a query on a name for which it is authoritative. Name servers providing answers for which they are not authoritative (for example, name servers for parent zones), do not set the AA bit.

Unbound can be setup to supply authoritative names for a local LAN. For example, if we wanted to hostname xbox360.home.lan to resolve to the private ip address 10.0.0.3 .

You might also want to have a look at our simple DNS Verify script. It will verify forward and reverse name resolution of you local LAN hostnames. This is a very quick way to make sure your DNS setup is correct.

Unbound DNS install

The first thing you need to do is install Unbound. We are using the latest release of **Unbound v1.4.9** for this example. Most modern operating systems have prebuilt packages (rpm, deb, tgz). You can always install Unbound from source if you want to. Either way you decide to install, you want to get the most recent version you can. Here are a few package manager lines just to help out:

```
## FreeBSD
$ cd /usr/ports/dns/unbound; make install clean
or
$ pkg install unbound

## OpenBSD
$ pkg_add -i unbound

## CentOS
$ yum install unbound

## Ubuntu
$ apt-get install unbound
```

Unbound DNS configuration examples

Simple recursive caching DNS (example 1)

This is the most simple, but fully functional Unbound example and a perfect solution for a small LAN with a few machines accessing the Internet. The configuration will query the root DNS servers for answers made by localhost or ips on the LAN at 10.0.0.0/16 and cache the results. The log level is at one(1) which will log any errors and print out statistics only when the Unbound daemon is shut down.

Notice that the forward-zone is commented out. You can use the forward-zone directive if you do _NOT_ wish to query the root DNS server and you want to use other resolving DNS servers. For example, we have OpenDNS.org and Google Public DNS configured here. You can replace those ips with the DNS servers of your ISP if you wanted to.

Simple recursive caching DNS setup and install

All you need to do is make sure Unbound is installed. Then place the following unbound.conf in place of your copy; i.e on the OpenBSD install it is located in /var/unbound/etc/unbound.conf . Then just start up the unbound daemon by typing "unbound". That's it.

```
## Simple recursive caching DNS
## unbound.conf -- https://DNSwatch.COM
#
server:
        interface: 0.0.0.0
        access-control: 10.0.0.0/16 allow
        verbosity: 1
```

```
#forward-zone:
#       name: "."
#       forward-addr: 8.8.8.8        # Google Public DNS
#       forward-addr: 216.87.84.211  # OpenNIC Colorado, US
#       forward-addr: 66.244.95.20   # OpenNIC Indiana, US
#       forward-addr: 72.14.189.120  # OpenNIC Texas, US
#       forward-addr: 4.2.2.4        # Level3 Verizon
```

Authoritative, validating, recursive caching DNS (example 2)

This example is a fully functional authoritative, validating, recursive caching DNS server for a local private LAN and very close to the one we personally use. Unbound will recursively query any hostname from the root DNS servers it does not have a cached copy of. It will validate the queries using DNSSEC and 0x20-encoded random bits to foil spoof attempts. Finally, this server will be the authoritative DNS for a few hostnames on our local private "home.lan" segment.

To get started there are few configuration steps to take. The first is to get a copy of the latest root DNS server listing called root.hints. The second is to get the DNSSEC root trusted key setup. Third, you need to setup any hostnames and ip address of your LAN. These steps are quite easy so lets do them first.

Pre setup

This example install is for OpenBSD. The install path is "/var/unbound/etc/" for all configuration files. You can place the following files anywhere as long as you tell Unbound where they reside. So, you can use this as an example for Ubuntu, RHEL and CentOS as well.

Step 1, root-hints: is the file which contains the listing of primary root DNS servers. Unbound does have a listing of root DNS servers in its code, but we want to make sure we have the most up to date copy. We normally update our copy once every six(6) months.

To query a hostname Unbound has to start at the top at the root DNS servers and work its way down to the authoritative servers (see the definition of a resolving DNS server above). Download a copy of the root hints from Internic and place it in the /var/unbound/etc/root.hints file. This file will be called by the root-hints: directive in the unbound.conf file.

```
fetch ftp://FTP.INTERNIC.NET/domain/named.cache -o /var/unbound/etc/root.hints
```

Step 2, auto-trust-anchor-file: which contains the key for the root server so DNSSEC can be validated. We need to tell Unbound that we trust the root server so it can start to develop a chain of trust down to the hostname we want resolved and validated using DNSSEC.

For this example, we create a file in "/var/unbound/etc/root.key" and put the following line in it. This is the 2010 trust anchor for the root zone. You can independently verify the root zone anchor by going to the IANA.org Index of /root-anchors.

```
. IN DS 19036 8 2 49AAC11D7B6F6446702E54A1607371607A1A41855200FD2CE1CDDE32F24E8FB5
```

Also make sure that the "/var/unbound/etc/root.key" is owned by the user the Unbound daemon is running as. Our user on the OpenBSD install is "_unbound". Then make sure the "_unbound" user can write to the file.

Step 3, locally served zones are the hostnames and ips of the local LAN you want Unbound to be authoritative for. Scroll down the unbound.conf and look for the local-zone, local-data and local-data-ptr directives. You just need to change these to match the names of the machines and ip addresses of your network.

In the example we have the hostname firewall.home.lan resolving to the ip address 10.0.0.1. We also have a reverse lookup allowing 10.0.0.1 to resolve back to the hostname firewall.home.lan. There are others just to give you a good idea of the format.

The pre setup is done. We now have a root DNS hints file of the primary root servers. We also have a trust anchor file of the root server so Unbound can create a chain of trust for DNSSEC. In the future as the root DNS key changes Unbound will automatically update the root.key file for

us. Lastly, we have a few hostnames and ips of LAN machines we want to authoritatively resolve.

Authoritative, validating, recursive caching DNS setup and install

All you need to do is make sure unbound is installed. Then place the following unbound.conf in place of your copy; i.e. on the OpenBSD install it is located in /var/unbound/etc/unbound.conf . Make sure the root hints and the trust anchor is in place as directed by the instructions above. Then just start up the unbound daemon by typing "unbound". That's it.

unbound.conf

```
## Authoritative, validating, recursive caching DNS
## unbound.conf -- https://DNSwatch.COM
#
server:
  # log verbosity
    verbosity: 1

  # specify the interfaces to answer queries from by ip-address.  The default
  # is to listen to localhost (127.0.0.1 and ::1).  specify 0.0.0.0 and ::0 to
  # bind to all available interfaces.  specify every interface[@port] on a new
  # 'interface:' labeled line.  The listen interfaces are not changed on
  # reload, only on restart.
    interface: 127.0.0.1

  # port to answer queries from
    port: 53

  # Enable IPv4, "yes" or "no".
    do-ip4: yes

  # Enable IPv6, "yes" or "no".
    do-ip6: no

  # Enable UDP, "yes" or "no".
    do-udp: yes

  # Enable TCP, "yes" or "no". If TCP is not needed, Unbound is actually
  # quicker to resolve as the functions related to TCP checks are not done.i
  # NOTE: you may need tcp enabled to get the DNSSEC results from *.edu domains
  # due to their size.
    do-tcp: yes

  # control which client ips are allowed to make (recursive) queries to this
  # server. Specify classless netblocks with /size and action.  By default
  # everything is refused, except for localhost.  Choose deny (drop message),
  # refuse (polite error reply), allow (recursive ok), allow_snoop (recursive
  # and nonrecursive ok)
    access-control: 127.0.0.0/8 allow
    access-control: 10.0.0.0/16 allow

  # Read  the  root  hints from this file. Default is nothing, using built in
  # hints for the IN class. The file has the format of  zone files,  with  root
  # nameserver  names  and  addresses  only. The default may become outdated,
  # when servers change,  therefore  it is good practice to use a root-hints
  # file.  get one from ftp://FTP.INTERNIC.NET/domain/named.cache
    root-hints: "/var/unbound/etc/root.hints"

  # enable to not answer id.server and hostname.bind queries.
    hide-identity: yes

  # enable to not answer version.server and version.bind queries.
    hide-version: yes

  # Will trust glue only if it is within the servers authority.
  # Harden against out of zone rrsets, to avoid spoofing attempts.
  # Hardening queries multiple name servers for the same data to make
  # spoofing significantly harder and does not mandate dnssec.
    harden-glue: yes

  # Require DNSSEC data for trust-anchored zones, if such data is absent, the
  # zone becomes  bogus.  Harden against receiving dnssec-stripped data. If you
  # turn it off, failing to validate dnskey data for a trustanchor will trigger
  # insecure mode for that zone (like without a trustanchor).  Default on,
  # which insists on dnssec data for trust-anchored zones.
    harden-dnssec-stripped: yes
```

```
  # Use 0x20-encoded random bits in the query to foil spoof attempts.
  # http://tools.ietf.org/html/draft-vixie-dnsext-dns0x20-00
  # While upper and lower case letters are allowed in domain names, no significance
  # is attached to the case. That is, two names with the same spelling but
  # different case are to be treated as if identical. This means dnswatch.com is the
  # same as DNSwatch.COM which is the same as DNSWATCH.COM.
    use-caps-for-id: yes

  # the time to live (TTL) value lower bound, in seconds. Default 0.
  # If more than an hour could easily give trouble due to stale data.
    cache-min-ttl: 3600

  # the time to live (TTL) value cap for RRsets and messages in the
  # cache. Items are not cached for longer. In seconds.
    cache-max-ttl: 86400

  # perform prefetching of close to expired message cache entries.  If a client
  # requests the dns lookup and the TTL of the cached hostname is going to
  # expire in less than 10% of its TTL, unbound will (1st) return the ip of the
  # host to the client and (2nd) pre-fetch the dns request from the remote dns
  # server. This method has been shown to increase the amount of cached hits by
  # local clients by 10% on average.
    prefetch: yes

  # number of threads to create. 1 disables threading. This should equal the number
  # of CPU cores in the machine. Our example machine has 4 CPU cores.
    num-threads: 4


  ## Unbound Optimization and Speed Tweaks ###

  # the number of slabs to use for cache and must be a power of 2 times the
  # number of num-threads set above. more slabs reduce lock contention, but
  # fragment memory usage.
    msg-cache-slabs: 8
    rrset-cache-slabs: 8
    infra-cache-slabs: 8
    key-cache-slabs: 8

  # Increase the memory size of the cache. Use roughly twice as much rrset cache
  # memory as you use msg cache memory. Due to malloc overhead, the total memory
  # usage is likely to rise to double (or 2.5x) the total cache memory. The test
  # box has 4gig of ram so 256meg for rrset allows a lot of room for cacheed objects.
    rrset-cache-size: 256m
    msg-cache-size: 128m

  # buffer size for UDP port 53 incoming (SO_RCVBUF socket option). This sets
  # the kernel buffer larger so that no messages are lost in spikes in the traffic.
    so-rcvbuf: 8m

  ## Unbound Optimization and Speed Tweaks ###


  # Enforce privacy of these addresses. Strips them away from answers.  It may
  # cause DNSSEC validation to additionally mark it as bogus.  Protects against
  # 'DNS Rebinding' (uses browser as network proxy).  Only 'private-domain' and
  # 'local-data' names are allowed to have these private addresses. No default.
    private-address: 10.0.0.0/8
    private-address: 172.16.0.0/12
    private-address: 10.0.0.0/16
    private-address: 192.254.0.0/16

  # Allow the domain (and its subdomains) to contain private addresses.
  # local-data statements are allowed to contain private addresses too.
    private-domain: "home.lan"

  # If nonzero, unwanted replies are not only reported in statistics, but also
  # a running total is kept per thread. If it reaches the threshold, a warning
  # is printed and a defensive action is taken, the cache is cleared to flush
  # potential poison out of it.  A suggested value is 10000000, the default is
  # 0 (turned off). We think 10K is a good value.
    unwanted-reply-threshold: 10000

  # IMPORTANT FOR TESTING: If you are testing and setup NSD or BIND  on
  # localhost you will want to allow the resolver to send queries to localhost.
  # Make sure to set do-not-query-localhost: yes . If yes, the above default
  # do-not-query-address entries are present.  if no, localhost can be queried
  # (for testing and debugging).
    do-not-query-localhost: no

  # File with trusted keys, kept up to date using RFC5011 probes, initial file
  # like trust-anchor-file, then it stores metadata.  Use several entries, one
  # per domain name, to track multiple zones. If you use forward-zone below to
```

```
  # query the Google DNS servers you MUST comment out this option or all DNS
  # queries will fail.
    auto-trust-anchor-file: "/var/unbound/etc/root.key"

  # Should additional section of secure message also be kept clean of unsecure
  # data. Useful to shield the users of this validator from potential bogus
  # data in the additional section. All unsigned data in the additional section
  # is removed from secure messages.
    val-clean-additional: yes

  # Blocking Ad Server domains. Google's AdSense, DoubleClick and Yahoo
  # account for a 70 percent share of all advertising traffic. Block them.
    local-zone: "doubleclick.net" redirect
    local-data: "doubleclick.net A 127.0.0.1"
    local-zone: "googlesyndication.com" redirect
    local-data: "googlesyndication.com A 127.0.0.1"
    local-zone: "googleadservices.com" redirect
    local-data: "googleadservices.com A 127.0.0.1"
    local-zone: "google-analytics.com" redirect
    local-data: "google-analytics.com A 127.0.0.1"
    local-zone: "ads.youtube.com" redirect
    local-data: "ads.youtube.com A 127.0.0.1"
    local-zone: "adserver.yahoo.com" redirect
    local-data: "adserver.yahoo.com A 127.0.0.1"

  # locally served zones can be configured for the machines on the LAN.

    local-zone: "home.lan." static

    local-data: "firewall.home.lan.  IN A 10.0.0.1"
    local-data: "laptop.home.lan.    IN A 10.0.0.2"
    local-data: "xbox360.home.lan.   IN A 10.0.0.3"
    local-data: "ps3.home.lan.       IN A 10.0.0.4"
    local-data: "dhcp5.home.lan.     IN A 10.0.0.5"

    local-data-ptr: "10.0.0.1  firewall.home.lan"
    local-data-ptr: "10.0.0.2  laptop.home.lan"
    local-data-ptr: "10.0.0.3  xbox360.home.lan"
    local-data-ptr: "10.0.0.4  ps3.home.lan"
    local-data-ptr: "10.0.0.5  dhcp5.home.lan"

  # Unbound can query your NSD or BIND server for private domain queries too.
  # On our NSD page we have NSD configured to serve the private domain,
  # "home.lan". Here we can tell Unbound to connect to the NSD server when it
  # needs to resolve a *.home.lan hostname or IP.
  #
  # private-domain: "home.lan"
  # local-zone: "0.0.10.in-addr.arpa." nodefault
  # stub-zone:
  #      name: "home.lan"
  #      stub-addr: 10.0.0.111@53

  # If you do not want to use the root DNS servers you can use the following
  # forward-zone to forward all queries to Google DNS, OpenDNS.com or your
  # local ISP's dns servers for example. If use use forward-zone you must make
  # sure to comment out the auto-trust-anchor-file directive above or else all
  # DNS queries will fail. We highly suggest using Google DNS as it is
  # extremely fast.
  #
    forward-zone:
       name: "."
       forward-addr: 8.8.8.8        # Google Public DNS
       forward-addr: 4.2.2.4        # Level3 Verizon
       forward-addr: 74.207.247.4   # OpenNIC DNS
#
#
## Authoritative, validating, recursive caching DNS
## unbound.conf -- https://DNSwatch.COM
```

NOTE: DNSwatch.COM offers a **OpenBSD Pf Firewall "how to" (pf.conf)** if you need it. We cover a HFSC scheduler example with a DNS priority queue integrated into a working pf.conf rule set.

Unbound DNS thoughts, ideas and theories

Unbound DNS cluster with BIND or NSD master server

Unbound is the perfect front line soldier for DNS queries from LAN clients. It is fast, reliable, stable and very secure. BIND (named) or NSD (Name Server Daemon) can be kept on the back end network to be an authoritative DNS to the Unbound cluster. This way you keep your primary DNS data segregated and unencumbered on the BIND or NSD server while the Unbound cluster servers do the resolving, caching and validation of zones for clients.

The idea is to have a few Unbound validating, recursive and caching DNS servers which LAN clients can query. Then use BIND (named) as an authoritative server which can resolve internal LAN names only. LAN clients will NEVER access the BIND DNS server and BIND will never go out to the Internet. BIND's only job is to serve internal names to the Unbound DNS server cluster. The Unbound cluster will serve all LAN clients. If Unbound needs to resolve a private ip it will ask the BIND server for ips and then cache the response. If the client needs an external ip, lets say from google.com or cnn.com, Unbound will recursively query the Internet root DNS servers and cache the response.

Here is a rough ASCII diagram of the setup. LAN Clients are all the internal machines of the local area network (LAN). The Internet DNS is basically any external DNS server. The entries for Unbound #1 through #3 are separate machines with Unbound running on them. The final machine is the Private BIND LAN DNS.

```
                INTERNET DNS
                      |
                      |
                -- Unbound #1 --
              /                      private authoritative only
LAN Clients -- --- Unbound #2 --  --   BIND or NSD dns server
                                /        ( 10.0.0.111 )
                -- Unbound #3 --
```

The configuration is quite easy. First take a copy of the Unbound example #2 called "Authoritative, validating, recursive caching DNS" from above. We will be adding directives to the bottom of that config. We just need to add the stub-zone directives for any internal names we want unbound to ask BIND about. Any zones not pointing to our authoritative server will go out to the root DNS servers for an answer. BTW, since BIND is going to be our authentication server you can comment out any of the local-zone, local-data and local-data-ptr directives.

Lets say our private NSD or BIND server is authoritative for this internal LAN domain:

- home.lan

We need to tell the Unbound cluster servers that if they are looking for the private home.lan domain to ask the authoritative DNS server, otherwise go check with the root DNS servers. The NSD server's ip is 10.0.0.111 as in the ASCII diagram.

Pay special attention to the stub-zone directive we are using. Stub-zone is only used to point queries to an authoritative server like a NSD dns server. The forward-zone directive can only be used to point queries to a resolving dns server like OpenDNS.com or you local ISP's caching server. The two are not interchangeable.

NOTE: We will be using the NSD configuration on the NSD (Name Server Daemon) Tutorial page with this example.

```
## Add this to the bottom of example #2's unbound.conf configuration
## Check out our NSD Tutorial at https://DNSwatch.COM/dns-docs/NSD/

   # This local-zone line will tell unbound that private addresses like
   # 10.0.0.0/8 can send queries to a stub zone authoritative server like NSD.
   local-zone: "10.in-addr.arpa." nodefault

   # This is the FORWARD lookup stub zone pointing to the NSD authoritative
   # server. When a client queries for firewall.home.lan the question is sent
   # to the NSD server located at 10.0.0.111 and NSD returns the answer
   # "10.0.0.1".
   stub-zone:
        name: "home.lan"
        stub-addr: 10.0.0.111

   # This is the REVERSE (rDNS) dns lookup for the home.lan zone. When a client
   # asks for the hostname belonging to the ip address 10.0.0.1 the NSD
   # authoritative server at 10.0.0.111 will send back the answer
```

```
   # "firewall.home.lan".
   stub-zone:
        name: "10.in-addr.arpa."
        stub-addr: 10.0.0.111
```

That's about it. A client asking for an internal dns hostname like, laptop.home.lan.lan will make Unbound query the NSD server (10.0.0.111); the answer will be cached by Unbound for later queries. Any other queries for external hostnames (DNSwatch.COM for example) from LAN clients will have Unbound go to Internet servers for the answer. Clients can now query the Unbound cluster for any hostnames they want and we do not have to worry about our primary NSD dns servers being abused or overloaded. This setup is mainly designed to segregate the authoritative server off by itself and keep the primary DNS configuration safe.

Dnsspoof or 'Split horizon' with Unbound DNS

Dnsspoof is the ability to reply with an ip address which is NOT normally associated with a hostname. For example, lets say you have web server accessible to the internal and external networks. The webserver is in a DMZ on the internal LAN. The public uses the external hostname webserver.example.com which resolves to the ip 111.222.333.444 which traverses a firewall to the DMZ. The problem is when your internal LAN clients would also like to use webserver.example.com . The internal clients traffic would have to go out the firewall, hit the router and then travel back into the firewall into the DMZ thus adding a lot of unnecessary traffic to the external connection. Wouldn't it be nice just to send internal clients directly to the internal DMZ ip address?

We can setup Unbound to spoof the dns query of LAN clients so instead of getting the external address for the external name, they receive an internal ip for the external name. webserver.example.com which normally resolves to 111.222.333.444 now resolves 10.0.0.222 . 10.0.0.222 would be the same web server which is located on the inside of the firewall in a DMZ.

```
## DnsSpoof of webserver.example.com (exact domain name match)
local-data: "webserver.example.com. IN A 10.0.0.222"
```

DnsSpoof'ing might also be useful to stop LAN clients from going to sites that might not be appropriate to go to or not allowed on your network. You can also use this method to stop your clients from going to ad servers. For example we do not want anyone going to facebook.com or doubleclick.net, or even any of its sub domains, instead giving them the ip address 10.0.0.111 of our local web server explaining our Internet terms of use.

```
## DnsSpoof of unwanted or restricted sites
local-zone: "doubleclick.net" redirect
local-data: "doubleclick.net A 10.0.0.111"

local-zone: "facebook.com" redirect
local-data: "facebook.com A 10.0.0.111"
```

The dns query by an internal client would then result in the following differences. Note that using the redirect method will now redirect all sub domains to 10.0.0.111 as well.

```
## Normal DNS resolution for facebook.com
$  host facebook.com
facebook.com has address 69.63.189.16
facebook.com has address 69.63.181.12
facebook.com has address 69.63.189.11
facebook.com mail is handled by 10 smtpin.mx.facebook.com.

## DnsSpoof of facebook.com
$  host facebook.com
facebook.com has address 10.0.0.111

## Sub domains are also redirected
$  host ads.facebook.com
ads.facebook.com has address 10.0.0.111
```

Questions?

Is the system time critical to Unbounds operation?

Yes. Unbound is very paranoid about its DNS records. If a record is out of date, or even worse, if the record exists in some future date then Unbound will not resolve that domain correctly.

A common problem is you have to reboot the system and the BIOS time is incorrect. Then Unbound is started on boot and uses the current incorrect system time. You then set the correct time manually. Now either all of your dns records are expired because the time was set to the past or the records are illegal since the time is set to the future.

Make sure that the system time of your machine is as correct as possible by using NTPD of a GPS time server. Then start Unbound when the time is correct.

What random number generator PRNG does Unbound use?

Unbound uses a cryptographic strength random number generator. The arc4random() generator from OpenBSD is used. This means that predicting the random numbers generated by unbound is equivalent to cracking an encryption cipher. The random number generator is seeded with entropy. Real entropy from the system /dev/random is used to seed the random number generator. Thus, the starting values of the random number generator cannot easily be predicted. The OpenBSD package install of Unbound uses /dev/arandom instead for a more random entropy and faster seed creation.

What is dns-0x20 capitalization randomization ?

Capitalization randomization is also called dns-0x20. This is an experimental resilience method which uses upper and lower case letters in the question hostname to obtain randomness. On average adding about 7 or 8 bits of entropy. This method currently has to be turned on by the dns admin manually, as it may result in maybe 0.4% of domains getting no answers due to no support on the authoritative server side. In our second example we enable the directive "**use-caps-for-id: yes**" for better security using dns-0x20.

All this means is that dnswatch.com is the same as DNSwatch.COM which is the same as DNSWATCH.COM. When Unbound sends a query to a remote server it sends the hostname string in random upper and lower characters. The remote server must resolve the hostname as if all the characters were lower case. The remote server must then send the query back to Unbound in the same random upper and lower characters that Unbound sent. If the characters of the hostname in the response are in the same format as the query then the dns-0x20 check is satisfied.

Attackers hoping to poison a Unbound DNS cache must therefore guess the mixed-case encoding of the query and the timing of the return dns answer in addition to all other fields required in a DNS poisoning attack. dns-0x20 increases the difficulty of the attack significantly.

How can I query a DNS server for its software name and version ?

Use the "dig" command. The first line will query the local dns server or resolver located at localhost (127.0.0.1). This example shows the response of Unbound version 1.4.7. The next few lines are some examples of other DNS servers. BTW, If you enable the directives in Unbound to hide the identity (hide-identity and hide-version) of the server as we do on our second example the response will be completely blank.

```
## Unbound version v1.4.7
dig +short version.bind chaos txt
"unbound 1.4.7"

## Verizon running Vantio v4.4.0.2
dig @151.197.0.38 +short version.bind chaos txt
"Nominum Vantio 4.4.0.2"

## Level 3 is more paranoid.
dig @4.2.2.1 +short version.bind chaos txt
"If you have a legitimate reason for requesting this
info, please contact hostmaster@Level3.net"
```

How can I easily watch DNS traffic in real time ?

Try a program call "dnstop". It is available though most package managers. Once installed just execute the binary with the argument of the interface you want to watch for DNS traffic. For example, we would execute "dnstop em0" to watch traffic on the external Intel based em0 interface. There a few pages of options in dnstop that can be accessed using the number keys. The author's page has some examples of some DnsTop screen output looks like.

Failover in /etc/resolv.conf is too slow. What can I do ?

The normal timeout to failover from one name server to another in the /etc/resolv.conf file is 5 seconds. This is a bit too long. You can use the "options" directive to set the timeout to 3 tenths of a second. This example also enables "rotate" to load balance the name server queries, sets the attempts to one to only try a name server once before going to the next and sets the threshold for the number of dots which must appear in a name before an initial absolute query will be made.

```
domain domain.lan
search domain.lan domain.lan2 domain.lan3 domain.lan4
nameserver 192.168.0.1
nameserver 192.168.0.2
options ndots:1 timeout:0.3 attempts:1 rotate
```

How can I properly benchmark Unbound?

This was a response on the unbound mail list from Wijngaards, one of the developers, to a user regarding performance testing Unbound and getting the proper results. We have included the thread in its entirety.

...Question:

> I am currently testing unbound as part of a current project however I'm seeing some wildly different results comparing unbound-control stats to what queryperf results show. For example, I've logged the output of total.num.queries to a file every second and calculated the average result which shows 9474 queries per second, queryperf shows 6945 qps. Both results are from the same test load.

...Answer:

> This may surprise you, but this is actually a frequently asked question for the unbound server from people that try to measure its speed. (Boast: this is because of its speed).
>
> I see two explanations. The first, and my experience with others doing benchmarks, is that unbound outperforms queryperf. The second is a buffer overrun.
>
> - Unbound outperforms queryperf. Thus, it sends back more than queryperf can handle. The measurement you got from queryperf is the speed of queryperf itself.
> - Buffer overruns can be seen with netstat -su. The options so-rcvbuf and so-sndbuf can help you stop the buffer overruns for unbound.
>
> For faster results, http://unbound.net/documentation/howto_optimise.html

...Question:

> What is the proper way to monitor and benchmark the queries per second on an unbound server?

...Answer:

> Since unbound outperforms your sender, you must make your sender stronger. Start by using its own computer for the sender (do not run it on the server itself, this server is busy with unbound, and you have configured num-threads to the number of cpus in /proc/cpuinfo so it uses all CPUs, right? Also you want to include the speed of the IP stack in the result). Then add another computer that runs queryperf (you can add up the result qps for the two computers that run queryperf). Add more computers until the qps no longer goes up, or goes down a bit (a denial of service on the unbound server). That is the measured speed. From Jan-Piet Mens (who wrote a book about it), I heard he ended up with a whole classroom of pcs running queryperf .

...Conclusion:

The servers I've been testing are Intel E4500s with 2 GB of RAM and Intel gigabit NICs, they are only on 100 mbit ports however and are routed to through Cisco IP SLA. The test client is on an entirely separate subnet and is an 8 core opteron box with an Intel NIC however I've also run queryperf from multiple sources at once using virtual machines.

The actual queries in my data file are captured from real DNS traffic in our data center, about 50% of the queries end up hitting the cache. I can see caching effects in my graphs as well.

We ended up tweaking the unbound.conf a bit and changed the following settings which more than doubled the performance going from 8.8k qps [queries per second] to over 20k.

```
# outgoing-range: 4096
outgoing-range: 32768

# so-rcvbuf: 0
so-rcvbuf: 32m

# msg-cache-size: 4m
msg-cache-size: 256m

num-queries-per-thread: 4096

# rrset-cache-size: 4m
rrset-cache-size: 256m

# infra-cache-numhosts: 10000
infra-cache-numhosts: 100000
```

What is DNS Cache Poisoning ?

Poisoning a DNS resolver refers to the act of inserting fake, often malicious data into the resolvers cache. This can cause website visitors to be redirected from the site (e.g. their banking site) they thought to visit to a different web site, for example a phishing site.

Is DnsSpoof also considered cache poisoning? In fact, it is. The difference is that poisoning is the act of inserting bad data maliciously and dnsspoof'ing is inserting averting data we know about. It is a fine line differentiation depending on who is querying the data.

Unbound implements a number of methods to add random bits to secure queries against malicious deflection. The most important means to add randomness is to vary the port numbers from which the question is asked, another means is to use a hack that randomizes unused bits in the query name. Unbound uses 16 bits for the port randomization. To be precise, about 60000 random ports, avoiding ports below 1024 and avoiding IANA allocated UDP ports to avoid system instability of the server. The port randomization uses the same random number generator as the ID. Unbound takes care that a randomly drawn port is used for one query. Thus every query gets a freshly random port number.

Real protection, where you are not subject to the whims of chance, is achieved by using DNSSEC. DNSSEC uses digital signatures to protect the data. With DNSSEC there is no chance of poisoning, independent of the number of random bits used. Unbound implements the DNSSEC standard as specified in the RFCs (RFC4034, RFC4035). Unbound can act as a validator and can thus check the digital signatures attached in replies. Of course, the domain name owner must have inserted these digital signatures in the first place. In the absence of DNSSEC, unbound attempts to provide very good security. Without digital signatures, randomization and filtering are currently the only options.