Shared Private Resolver Operators' Implementation Guidelines

DNS Security and Privacy

1. **Practice 1:** DNSSEC validation **MUST** be enabled for recursive resolvers.

   DNSSEC validation may already be enabled by default if you've installed or upgraded your recursive resolver software package in recent years. For DNSSEC validation to work, a trust-anchor is required - that is, a copy of the root zone's public KSK. It may already be included in the package when you install the software, or, more likely, it will be fetched automatically at startup (this is the case for most DNS validators these days).

   To test that your recursive DNS resolver is actually performing validation, you can try the following:

   ```
   $ dig @ip.of.your.validator www.icann.org. SOA
   ```

   Check if the 'ad' bit is set in the 'flags' section of the response. If it is, then your resolver is performing DNSSEC validation.

   Implementation:

   BIND:

   If you're upgrading from an earlier version, check the options settings in your `named.conf` configuration file, and make sure the following settings are present:

   ```
   allow-recursion yes;
   dnssec-validation auto;
   ```

   The `'dnssec-validation auto;'` setting ensures that a root trust anchor is fetched and installed automatically.

   Additionally, you may want to enable additional logging for DNSSEC validation. This could be done using the following configuration snippet in BIND:

   ```
   logging {
           channel dnssec {
               file "/var/log/bind/dnssec" versions 3 size
   10M;
               print-time yes;
               severity info;
   ```

```
        };

        category dnssec { dnssec; };
};
```

Windows DNS:

In Windows DNS server for versions of Windows 2019 and newer, DNSSEC validation is said to be enabled, but this could not be verified at the time of this writing. To see what the status of DNSSEC validation is on your server, you can run the following command in PowerShell:

```
Get-DnsServerSetting
```

This will return a list of configured settings on your DNS server, including “EnableDnsSec”. If this is set to “True” then your Windows DNS service is already doing validation. If not, you may need to perform the following changes, as specified at https://blog.cdemi.io/enabling-windows-dns-server-to-validate-dnssec/

```
DnsCmd.exe /RetrieveRootTrustAnchors
```

You should see: “The root trust anchors were successfully retrieved…” and some more information about them becoming active during an refresh. You should be able to restart the DNS service, and see if the resolver starts performing validation.

It may also be necessary to do:

```
DnsCmd.exe /Config /enablednssec 1
```

PowerDNS resolver:

As documented at https://doc.powerdns.com/recursor/dnssec.html, PowerDNS Recursor has several modes of operation. The default setting is, in recent versions of PowerDNS Recursor, “process”, which means the validator will try and validate the query if DO (DNSSEC OK) or AD (Authenticated Data) is set in the query. To change this to an always-validate behavior, change the dnssec setting to ‘validate’. This is best for stub resolver (clients) that do not perform their own DNSSEC validation. Clients that wish to obtain unvalidated data can still do so using the CD (Check Disabled) bit (dig +cd).

PowerDNS Recursor ships with the root trust anchor already preconfigured.

Unbound:

DNSSEC validation is already enabled on Unbound. This is controlled using the

"`module-config`" configuration statement in `unbound.conf`, which is by default set to "`validator iterator`".

In addition to this, the following statement should be present:

```
auto-trust-anchor-file: "/var/lib/unbound/root.key"
```

This ensures that, upon start, Unbound will fetch a trust anchor, verify the signatures on it, install it into the location given, and use it to validate the signatures.

2. **Practice 2:** ACL statements **MUST** be used to restrict who may send recursive queries to your DNS resolvers/validators.

   When offering recursive DNS service, it is critical that you do not unwittingly open it up to third-party users outside the network.

   This requirement can be implemented at the network level (IP access lists/rules on the border router or the host OS), and at the service level (DNS service configuration). To facilitate auditing, it is recommended that ACL configuration be done as close to the service being offered, i.e.: on the service itself, although ultimately it's an operator's choice how they choose to implement this - if implementing at the network level, it will be up to each operator to consult with their network equipment/OS vendor on how to do this.

   When building your list of client subnets, remember to include your own internal networks and all customer subnets/ranges that you are announcing/routing to the Internet. Larger access providers may wish to operate distinct resolver infrastructures for different types of customers (private, commercial, mobile, broadband, etc.).

   Implementation:

   BIND:

   In BIND, one example of how to do this is as follows:

```
acl corp_net {
  192.168.1.0/24;
};
acl dsl_cust {
  192.168.2.0/24;
};
options {
  recursion yes;
  allow-recursion {
```

```
        corp_net;
        dsl_cust;
        127.0.0.0/8; }
};
```

PowerDNS resolver:

In PowerDNS, access to the resolver is controlled by the ‘allow-from’ setting. It defaults to 127.0.0.0/8, 10.0.0.0/8, 100.64.0.0/10, 169.254.0.0/16, 192.168.0.0/16, 172.16.0.0/12, ::1/128, fc00::/7, fe80::/10

Add any of your subnets, as in the BIND example above, to the ‘allow-from’ setting.

You can also read those from a file using the ‘allow-from-file` directive, one subnet per line. Note that this overrides anything in allow-from. See more at https://doc.powerdns.com/recursor/settings.html#allow-from

Unbound:

In the Unbound resolver, the ‘access-control’ setting in the ‘server:’ section of the configuration controls which clients are allowed to query the server. Example:

```
server:
  access-control: 10.0.0.0/8 allow
  access-control: 2001:DB8::/64 allow
```

Windows DNS:

Starting in Windows 2016, it should be possible to set up a policy restricting recursion to client subnets, according to https://docs.microsoft.com/en-us/windows-server/networking/dns/deploy/dns-policies-overview - but we haven’t tested that.

3. **Practice 3:** QNAME minimization **MUST** be enabled to mitigate leakage of domain names.

   QNAME minimization is normally the default in modern resolver software; however, be sure to check that it is enabled.

   Implementation:

   BIND:

Recent versions of BIND ship with QNAME minimization enabled, but in so-called “relaxed” mode. This causes BIND to fall back to not following RFC 9156 when it receives NXDOMAIN, SERVFAIL, or another unexpected answer to a minimal query. It is recommended to set the mode for minimization to “strict” nowadays, but consider switching to “relaxed” if problems arise.

```
options {
  qname-minimization strict;
};
```

PowerDNS resolver:

QNAME minimization is enabled by default in PowerDNS Recursor. It is controlled by the ‘qname-minimization’ parameter.

Unbound:

QNAME minimization is enabled in Unbound - controlled by the ‘qname-minimisation’ parameter (note the difference in the use of US and UK English for ‘minimisation/minimization’ in different implementations!).

```
server:
  qname-minimisation: yes
```

Windows DNS:

There appears to be no support for QNAME minimization in the Windows DNS recursive implementation, unfortunately.

DNS Availability and Resilience

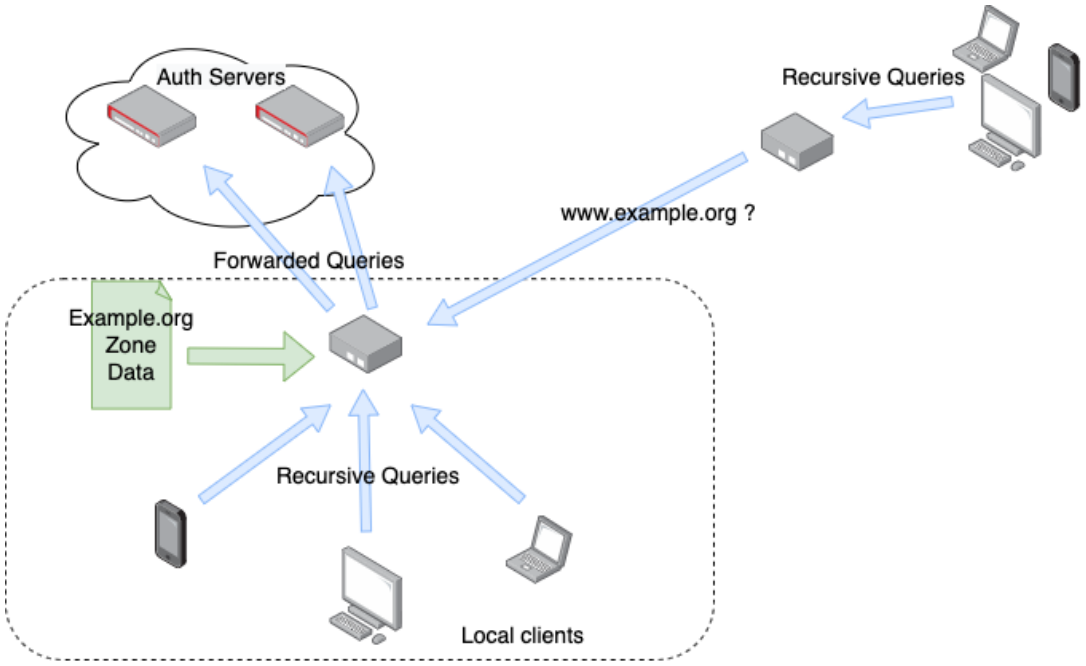
4. **Practice 4**: Authoritative and recursive DNS service **MUST NOT** coexist on the same DNS server.

   General Implementation Considerations:

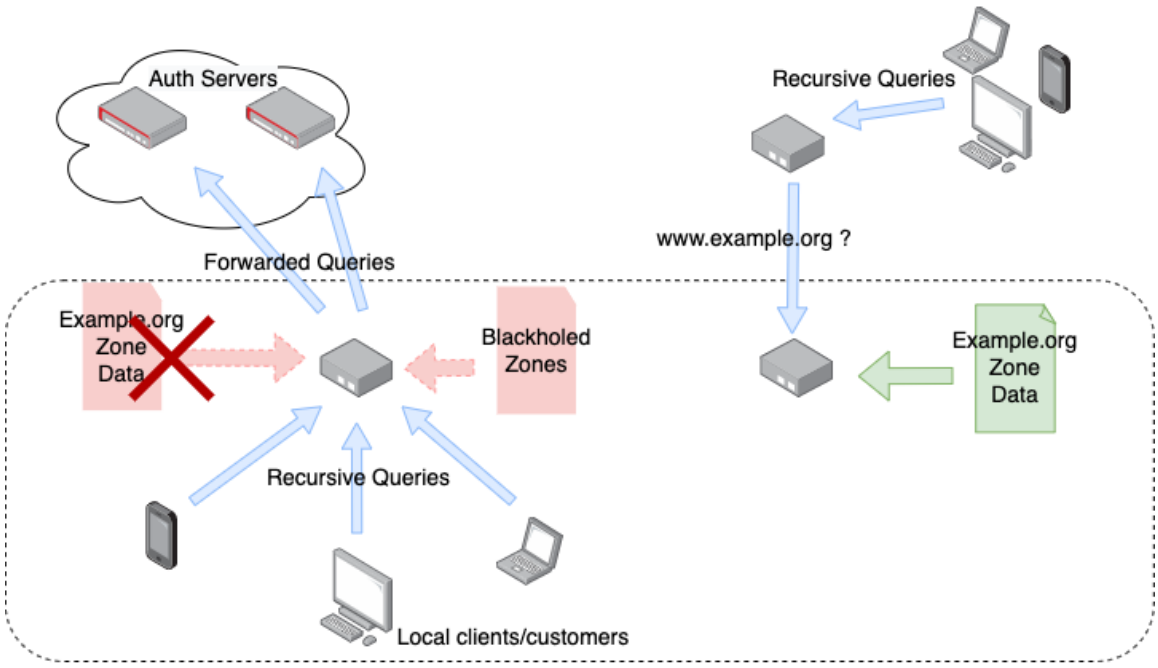
   Dedicated DNS recursive resolvers **should** be set up separately from the authoritative nameservers. Ideally, the recursive name servers will not be reachable from the wider Internet (see the corresponding BCPs on Network and Service security using ACLs).

   In the context of recursive servers, this means you should not configure them to also serve public authoritative zones even if the software allows it. An exception can be made for zones used to blackhole queries that should not be forwarded to the internet at large (such as RFC 1918 reverse lookup zones such as those managed by the AS112 project

- see https://www.as112.net/). The diagram below illustrates how a server configured to run as both authoritative and recursive is reconfigured, with the authoritative DNS server being split out as a distinct service.



Nameserver acting as both Recursive and Authoritative



Separation of duties - dedicated Recursive resolver and Authoritative servers

5. **Practice 5:** Your recursion services **MUST** have resilience by using at least two distinct servers that take diversity into consideration.

   On LAN environments, client network configuration is typically done using DHCP. Besides an IP address/netmask and default gateways, clients can be provided with a number of additional parameters/settings, including the DNS recursive resolvers to be used. There are similar mechanisms for 4G with PCO (Protocol Configuration Options) and IPCP for PPPoE and related technologies.

   General Implementation Considerations:

   At least two distinct servers must be given out to clients: even if the primary resolver is made redundant (via a load balancer or anycast, for instance), routing issues can make this primary instance unreachable. One option is to use a public open resolver as the secondary resolver, for instance 8.8.8.8, 1.1.1.1, 9.9.9.9, etc. (and corresponding IPv6 resolvers if IPv6 is configured). Alternatively, a secondary resolver (backup), possibly running on a different software platform, can be provided. If this is an option, consider placing the secondary/backup at a different PoP/site.

   There are no vendor-specific implementation guidelines, as it will depend on the access medium and equipment/provisioning platform manufacturer.

6. **Practice 6:** Monitoring of the services, servers, and network equipment that make up your DNS infrastructure **MUST** be implemented.

   General Implementation Considerations:

   Examples of resources and services that should be monitored:

   - Availability: does the DNS service answer?
     - Example: On port 53 UDP and TCP, does the DNS server return data if queried?
   - Correctness: does the DNS service return the expected data ?
     - Example: Query for a name and resource record type (for instance: www.example.org A, or example.org SOA), then check the result against a known good value
   - Latency: does the service respond in a timely fashion ?
     - Example: How long does it take for the service to respond to the above checks ? It should be within a reasonable timeframe, say less than 5ms for most authoritative queries, and probably less than 200-300ms for recursive queries. Account for the time it takes to fetch an answer if not already in the cache, and the network latency (round trip) between the monitoring service and the DNS service you are testing.

The above three tests can be performed as a single check using most monitoring platforms / services.

Example with the Nagios plugin, check_dig:

```
check_dig -4 -H a.icann-servers.net -l www.icann.org -w 2 -c 5 -a \
    www.vip.icann.org
```

The queried server is a.icann-servers.net, using IPv4 ('-4'). The queried name is 'www.icann.org'. The settings -w 2 and -c 5 set a warning and critical threshold if the server hasn't responded before 2 and 5 seconds have elapsed, respectively. `-a` is the expected result, in this case 'www.vip.icann.org'. We don't set a queried record type (-T), so it defaults to A.

Example output:

```
DNS OK - 0.123 seconds response time (www.icann.org.  3600 IN CNAME
www.vip.icann.org.)|time=0.122713s;2.000000;5.000000;0.000000
```

The same techniques apply for monitoring the availability and reachability of intermediate network devices - ICMP checks are often enough in most cases to detect failure of a router or switch in front of a DNS server.

If monitoring with a third-party service, there are many online providers which provide remote monitoring of availability and reachability (providing the service is using public IP addresses and filtering allows for remote monitoring).

Bonus Practice: *If you already have all of these practices in place, here is an additional recommended practice above and beyond the minimum KINDNS requirements.*

7. Practice 7 (**Privacy consideration**): DoT (DNS-over-TLS) or DoH (DNS-over-HTTPS) **SHOULD** be enabled.

   Deploying either is the easiest way to protect against eavesdropping and manipulation of DNS queries and man-in-the-middle attacks by encrypting DNS queries between stub and recursive resolvers, or between a forwarding and recursive resolver. Note that in the implementation notes below, we show how to enable DoH/DoT services on the server side. There is work in progress to automate discovery of DoH/DoT for outgoing queries, but this is not yet finalized (more on this at https://blog.powerdns.com/2022/06/13/probing-dot-support-of-authoritative-servers-just-try-it/).

   Implementation:

BIND:

See https://www.isc.org/blogs/bind-implements-doh-2021/, https://www.isc.org/blogs/doh-talkdns/, and https://bind9.readthedocs.io/en/latest/reference.html?highlight=DoH#interfaces

The following is a sample configuration (using BIND's native TLS service - TLS offloading is outside the scope of this document):

```
tls my-tls-srv {
  key-file "/etc/bind/doh.local.key";
  cert-file "/etc/bind/doh.local.crt";
  ciphers  \
    "HIGH:!kRSA:!aNULL:!eNULL:!RC4:!3DES:!MD5:!EXP:
    !PSK:!SRP:!DSS:!SHA1:!SHA256:!SHA384";
  prefer-server-ciphers yes;
};

http my-http-srv {
  endpoints {
    "/dns-query"; # this is default
  };
};

options {
  # other options...
  allow-recursion { ...; };
  tls-port 853;   # default
  https-port 443; # default

   listen-on port 53 { any; };     # listen on TCP/UDP 53
IPv4
  listen-on-v6 port 53 { any; }; # and on IPv6 as well
   listen-on port 443 tls my-tls-srv http my-http-srv {
any; };
   listen-on-v6 port 443 tls my-tls-srv http my-http-srv {
any; };
  listen-on port 853 tls my-tls-srv { any; };
  listen-on-v6 port 853 tls my-tls-srv { any; };
};
```

After testing that the configuration works with 'named-checkconf', restart BIND and verify it is now listening on ports 53, 443, and 853:

```
$ sudo netstat -apn | egrep ':(53|443|853)'|sort -u
```

Now you can test DoH:

```
$ dig +https @127.0.0.1 icann.org

...
;; SERVER: 127.0.0.1#443(127.0.0.1) (HTTPS)
```

Or you can test DoT:

```
$ dig +tls @127.0.0.1 icann.org
```

The output should resemble this:

```
...
;; SERVER: 127.0.0.1#443(127.0.0.1) (TLS)
...
```

PowerDNS Recursor:

PowerDNS Recursor doesn't directly support DoH and DoT. Instead, it relies on dnsdist, also developed by PowerDNS (OpenExchange), to be set up as a proxy/balancer in front of PowerDNS. There are more details here: https://dnsdist.org/guides/dns-over-https.html and https://dnsdist.org/guides/dns-over-tls.html

A general example is given here: https://blog.apnic.net/2020/02/28/how-to-deploy-dot-and-doh-with-dnsdist/. Note in this example that the backend recursive server could be any server such as PowerDNS, BIND9, etc.

Unbound:

Unbound has support for DoH and DoT natively, as long as it has been compiled with libnghttp2. This is the case in most modern distributions, and configuration is rather straightforward. See https://unbound.docs.nlnetlabs.nl/en/latest/topics/privacy/dns-over-https.html#using-doh

In the server section:

```
server:
    interface: ::@53
    interface: 0.0.0.0@53
```

```
interface: ::@443
interface: 0.0.0.0@443

interface: ::@853
interface: 0.0.0.0@853

tls-service-key: "unbound_server.key"
tls-service-pem: "unbound_server.pem"
tls-port: 853
https-port: 443
```

Now you can test DoH:

```
$ dig +https @127.0.0.1 icann.org

...
;; SERVER: 127.0.0.1#443(127.0.0.1) (HTTPS)
```

And test DoT:

```
$ dig +tls @127.0.0.1 icann.org
```

The output should resemble this:

```
...
;; SERVER: 127.0.0.1#443(127.0.0.1) (TLS)
...
```

Windows DNS:

Windows Server DNS unfortunately doesn’t offer DoH or DoT.