

Living on the Edge

*Greatly needed stub resolver capabilities
for applications and systems with the ~~getdns~~ library*



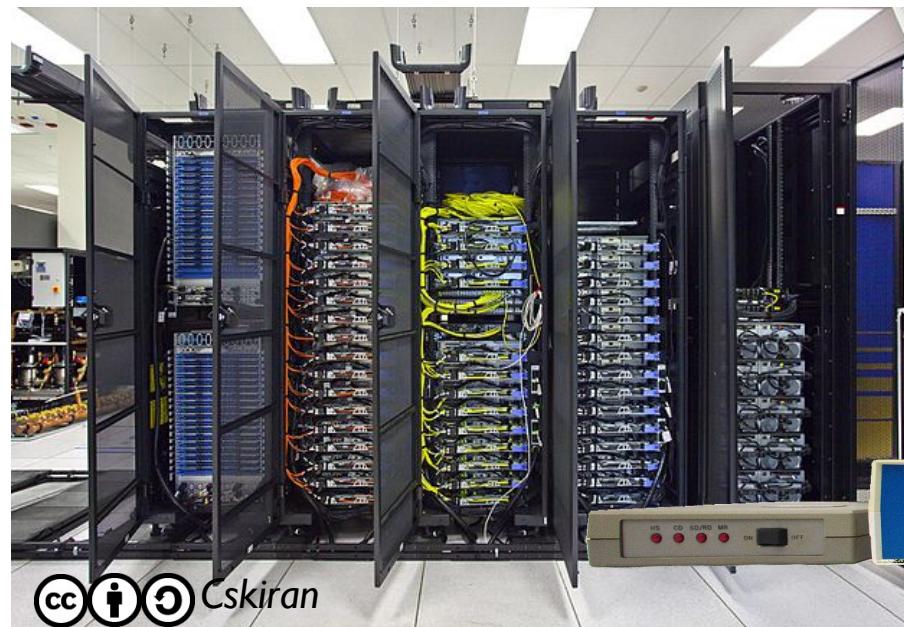
Willem Toorop
4 February 2018
FOSDEM18 (Bruxelles)

the edge (end user devices)



Muhammad Rafizeldi

the infrastructure



Cskiran



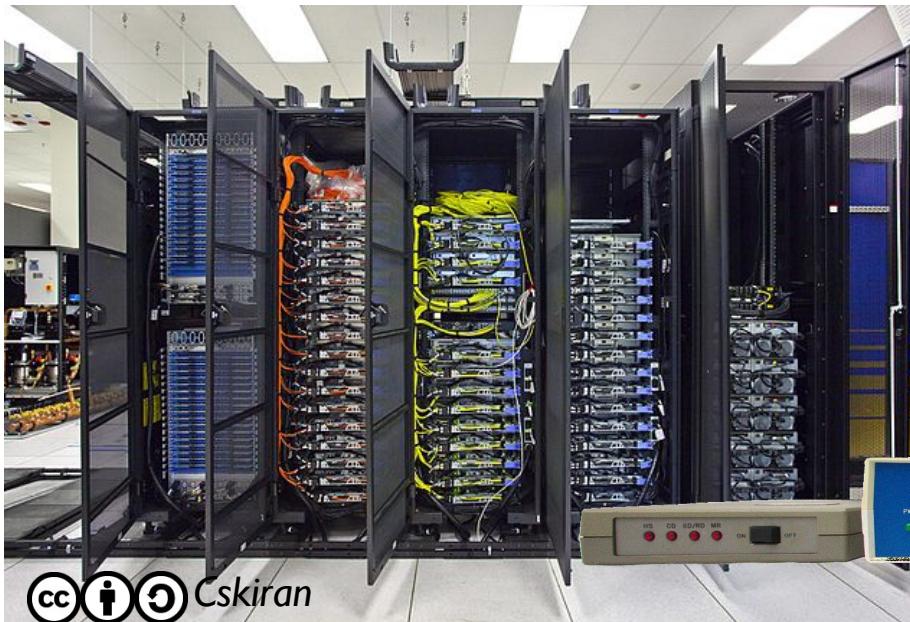
Pratyeka



cc i G Muhammad Rafizeldi



Encryption everywhere

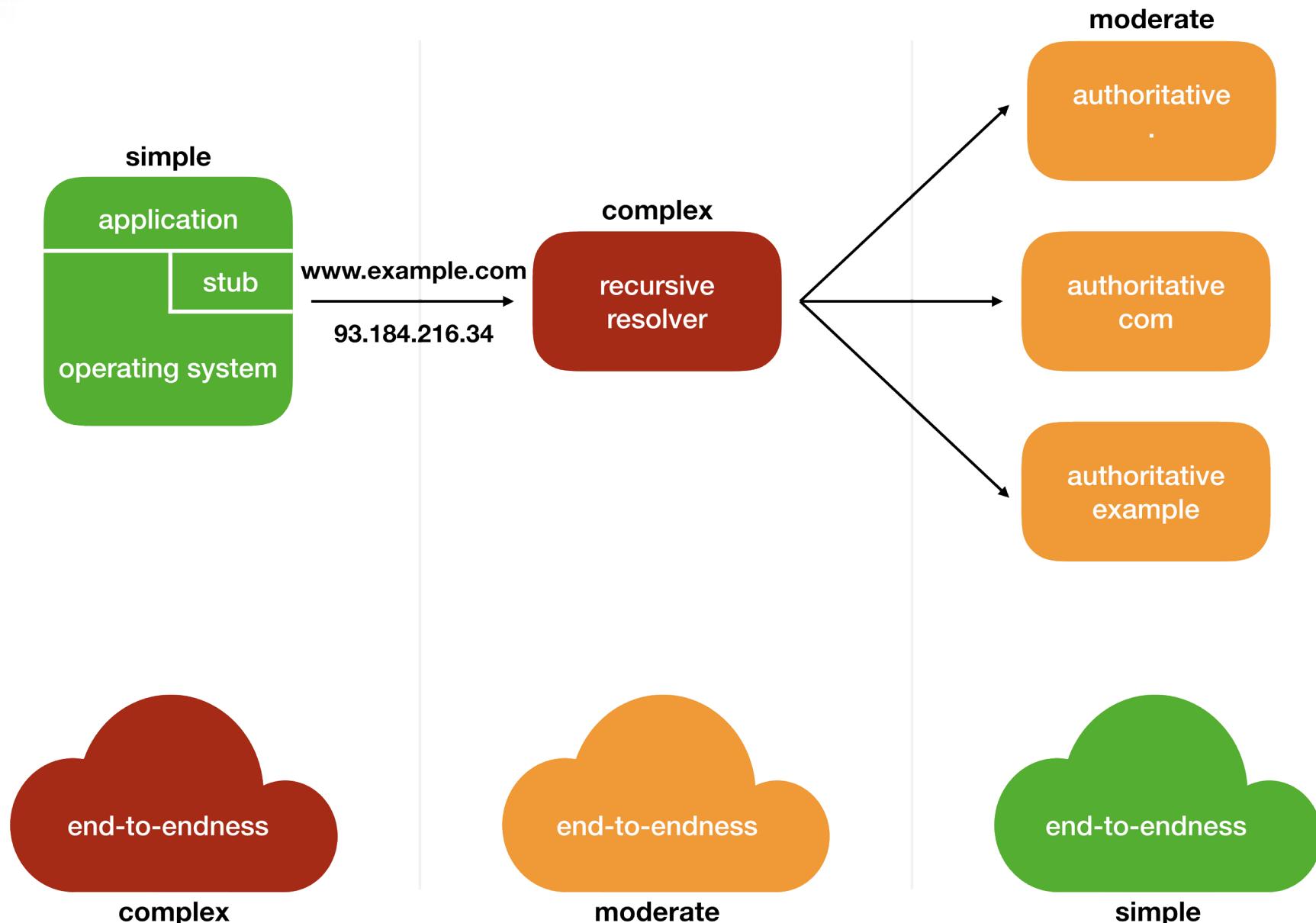


cc i G Cskiran

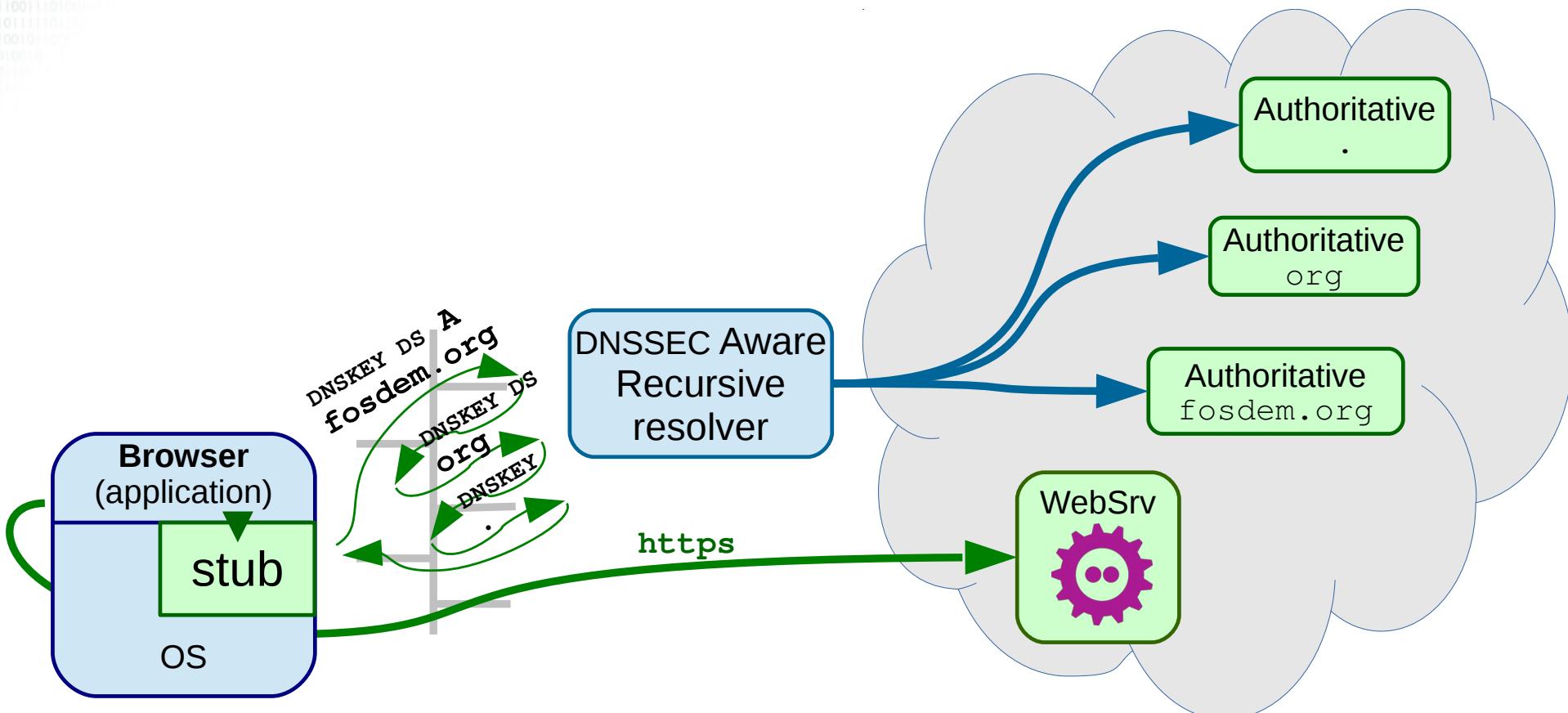


cc i G Pratyeka

The DNS ecosystem

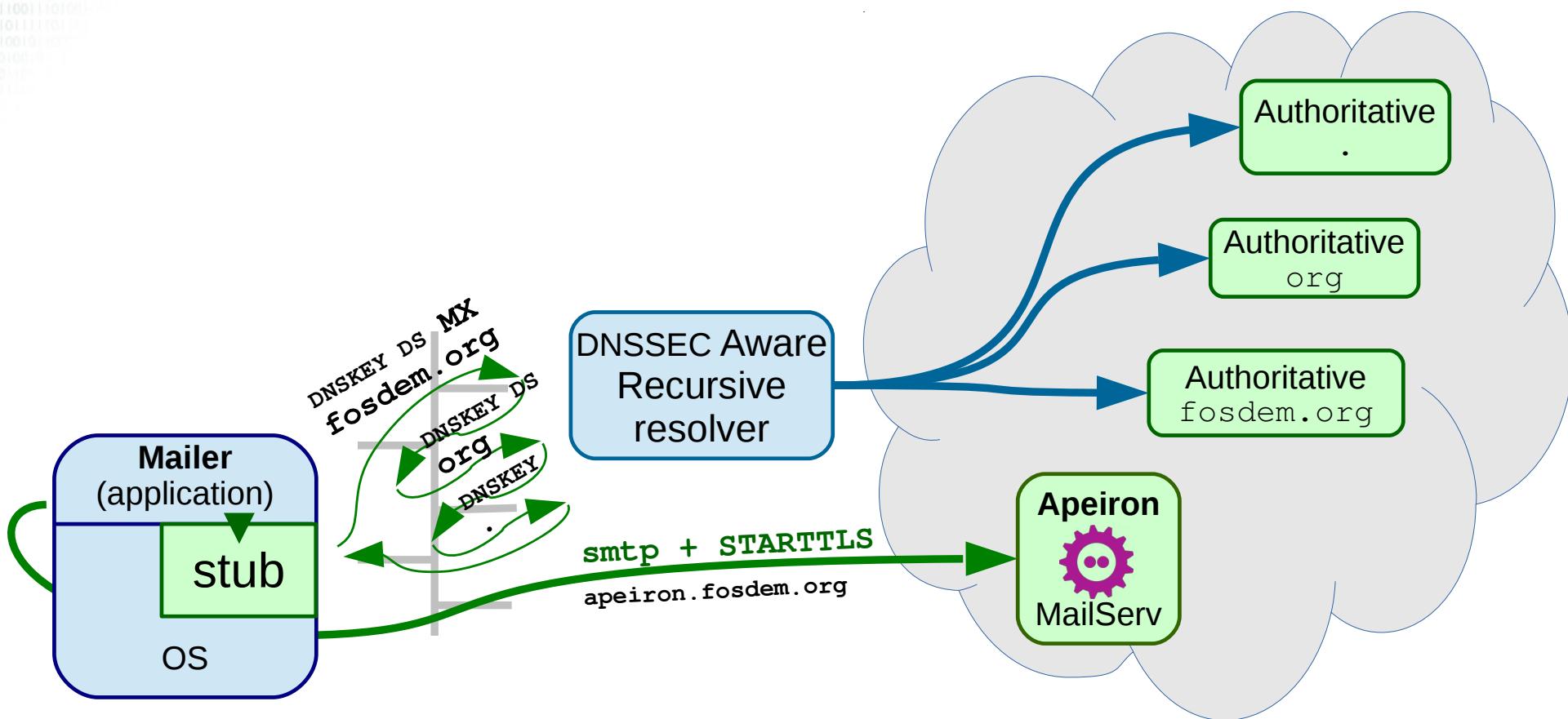


Encryption everywhere



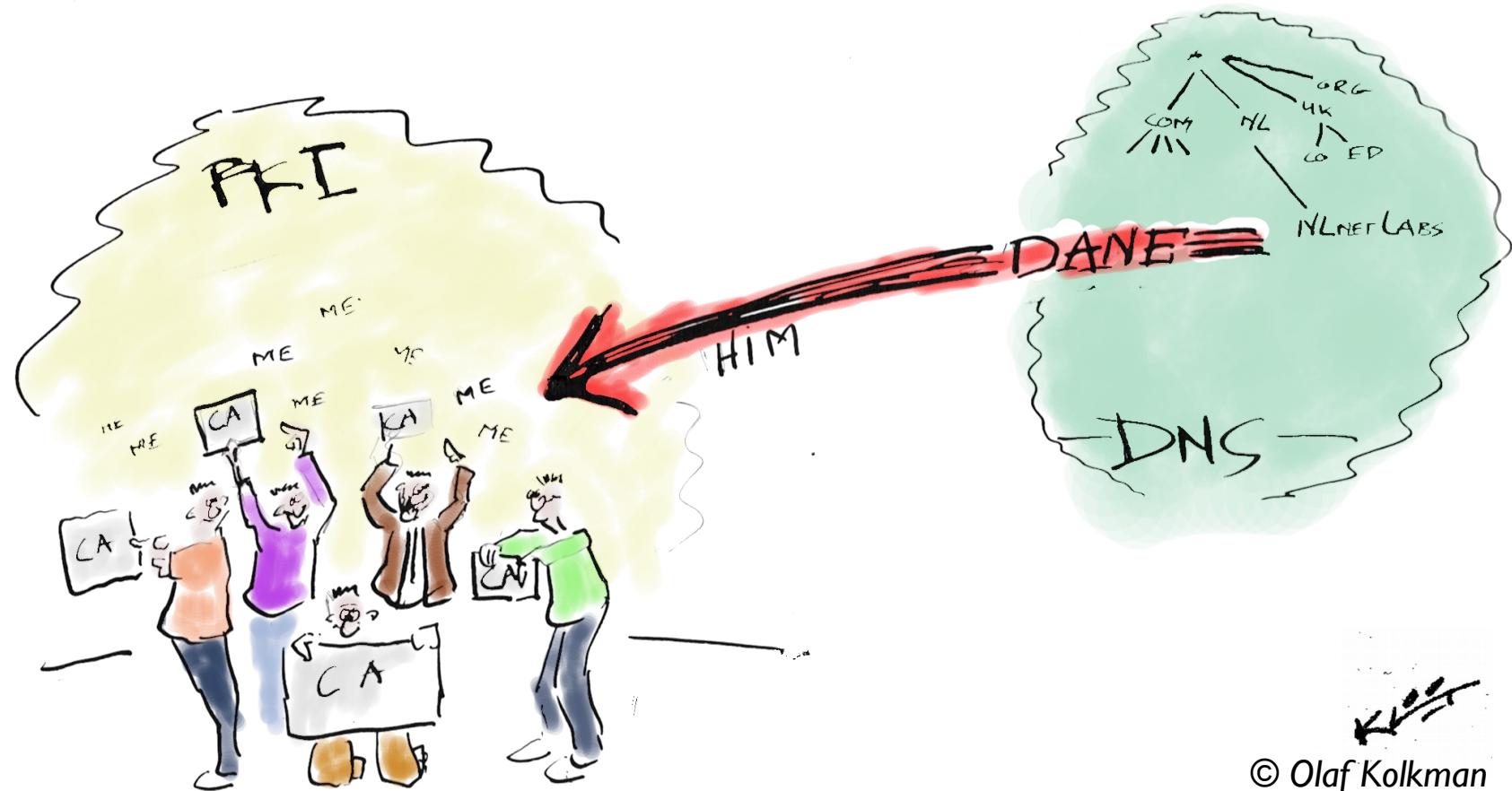
- TLS authenticates the DNS name

Encryption everywhere



- TLS authenticates the DNS name
- Still need DNSSEC for redirections *(MX, SRV)*

Encryption everywhere



- 1482 Certificate Authorities in 2010
Eckersley, Peter, and Jesse Burns. "An observatory for the SSLiverse." Talk at Defcon 18 (2010).
- Strengthen TLS with stub: DANE
- Signalling TLS support (for SMTP and the like)

Encryption everywhere

Ist hurdle: Trust Anchor Management

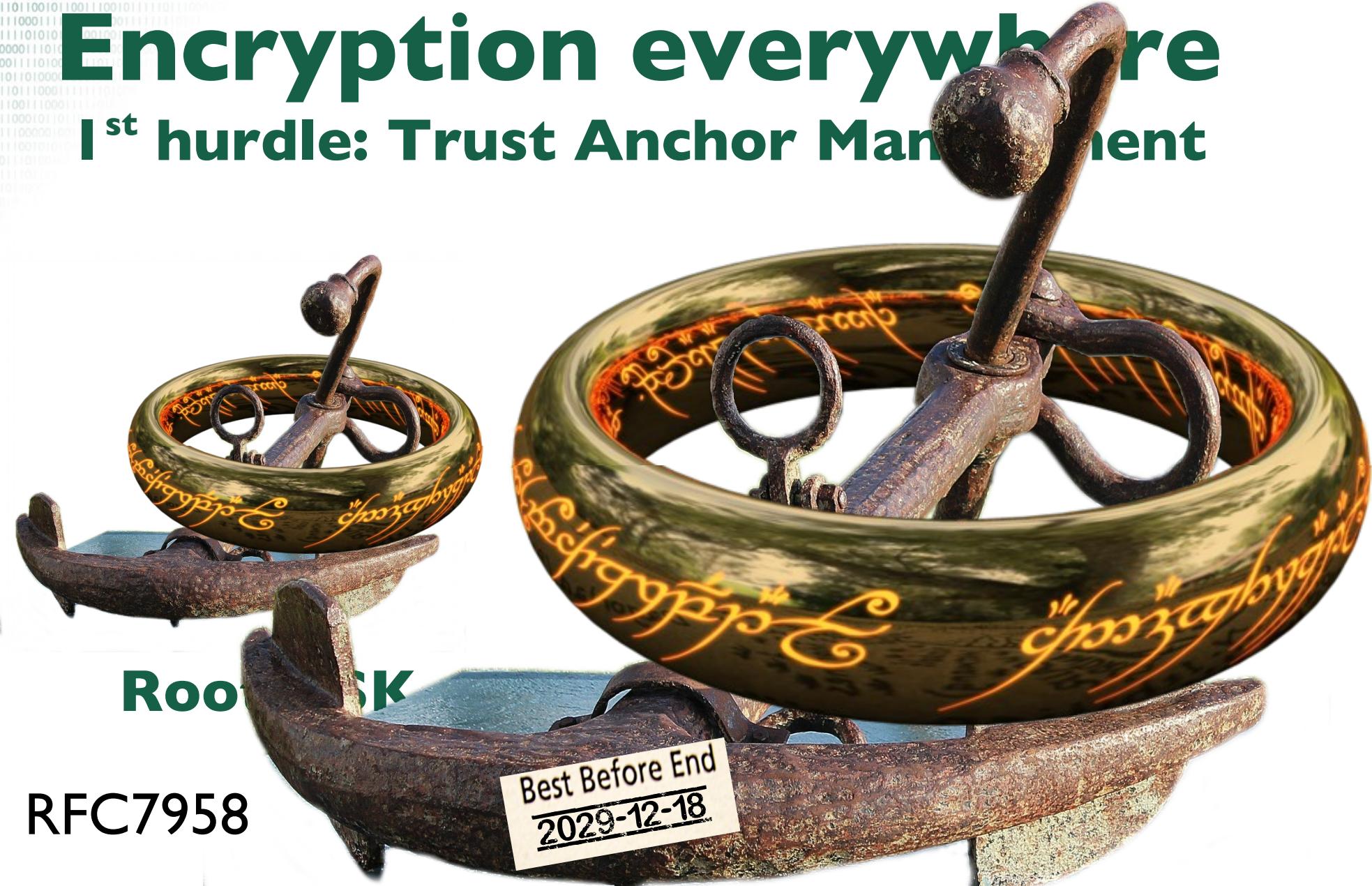


Root KSK

- RFC5011 (in protocol rollover)
- Assumes permanent running process with system privileges

Encryption everywhere

1st hurdle: Trust Anchor Management



- RFC7958

ICANN Root CA

En
I st h

Index of /root-anchors - Chromium



Name	Last modified	Size
Parent Directory		-
old/	2017-02-03 00:00	-
checksums-sha256.txt	2017-02-03 00:00	248
icannbundle.pem	2017-02-03 00:00	13K
root-anchors.p7s	2017-02-03 00:00	4.0K
root-anchors.xml	2017-02-03 00:00	651

RFC 7958

Root Zone Trust Anchor Publication

August 2016

It is important to note that the ICANN CA is not a DNSSEC trust anchor. Instead, it is an optional mechanism for verifying the content and origin of the XML and certificate trust anchors.

Abley, et al.

Informational

[Page 10]

Zero configuration DNSSEC

Fetch TA a la RFC7958, when:

- No trust anchor was present
- root DNSKEY did not validate

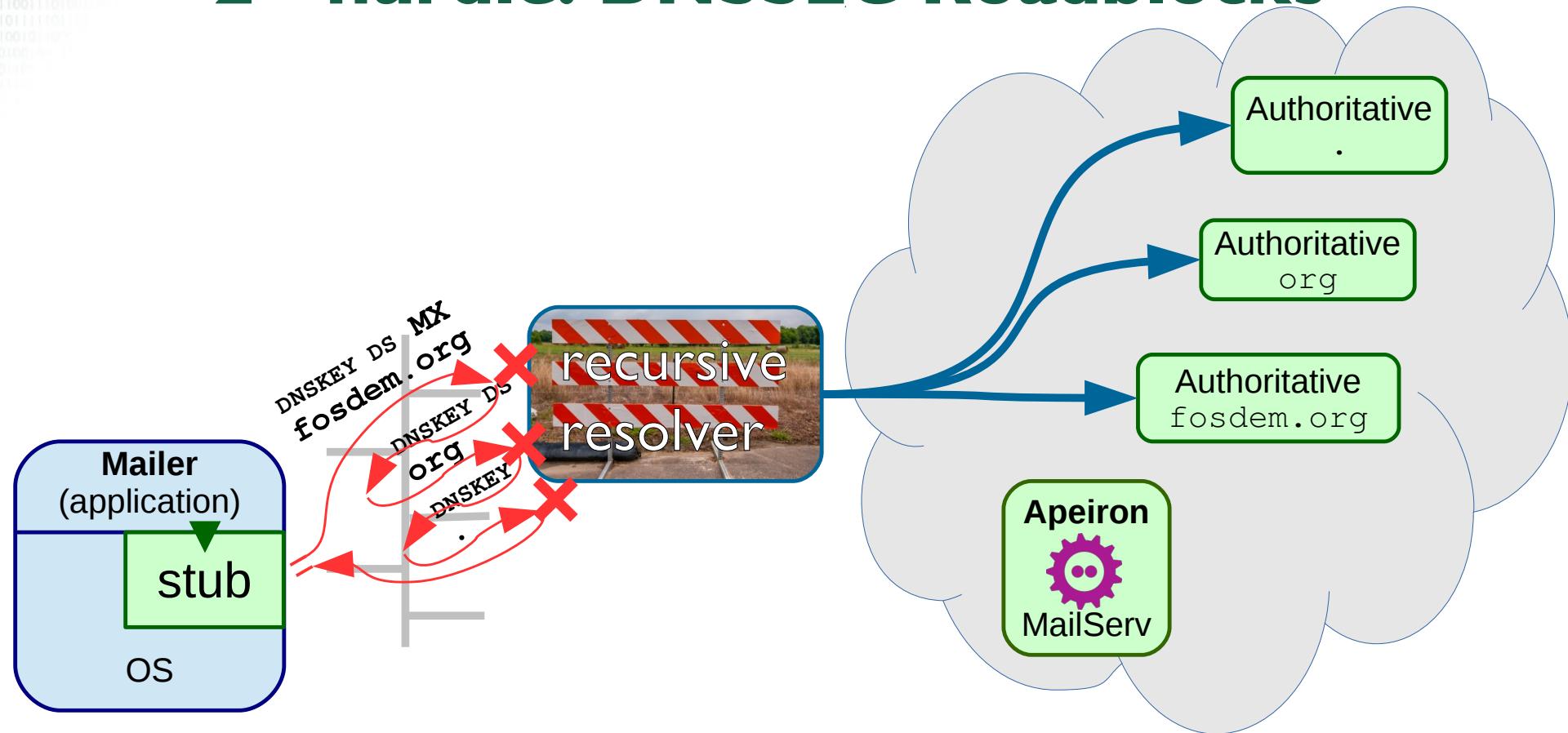
Track Root DNSKEY RRset



```
willem@makaak: ~
willem@makaak:~$ ls -l ~/.getdns
totaal 14
-rw----- 1 willem willem 4095 dec 13 12:47 root-anchors.p7s
-rw----- 1 willem willem 651 dec 13 12:47 root-anchors.xml
-rw----- 1 willem willem 1663 jan 23 14:51 root.key
willem@makaak:~$ █
```

Encryption everywhere

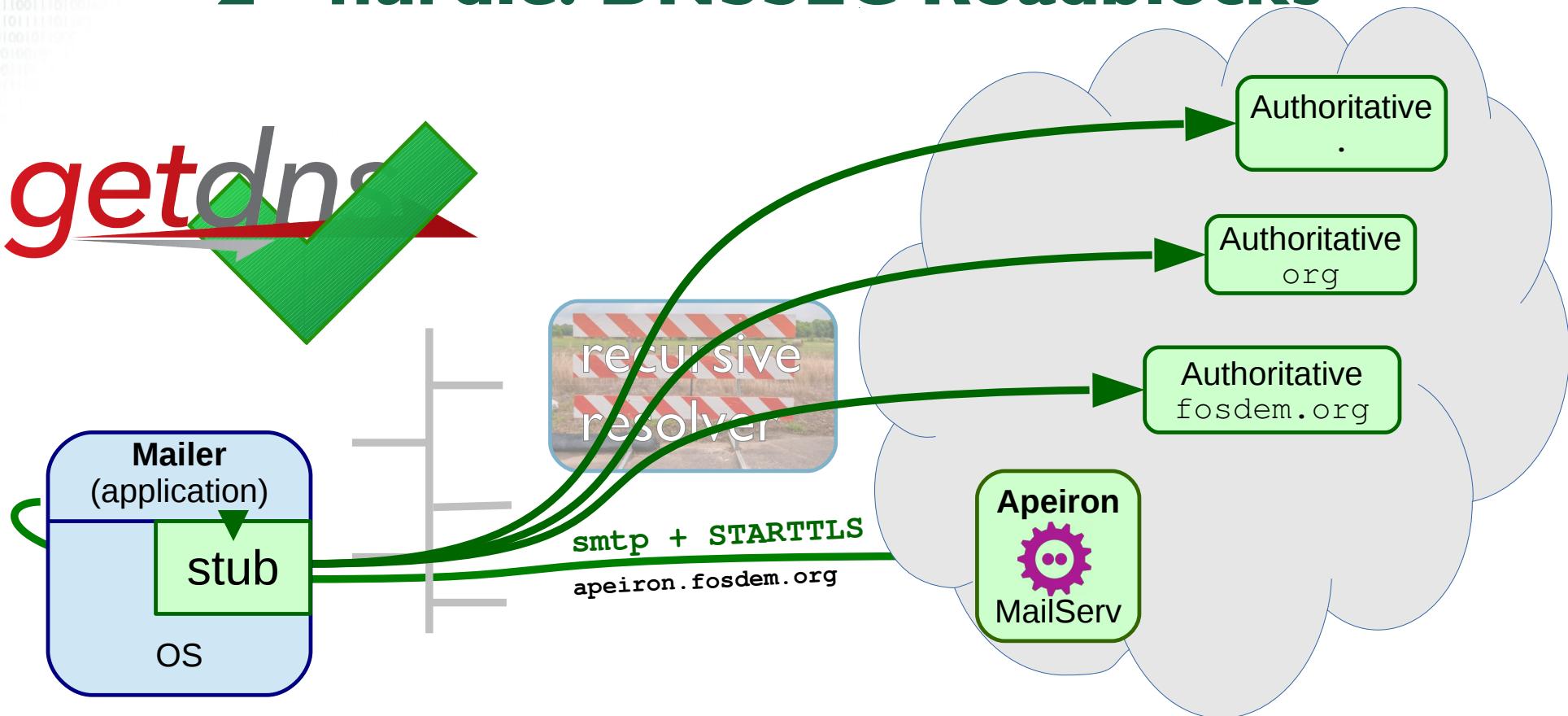
2nd hurdle: DNSSEC Roadblocks



- A DNSSEC Aware recursive resolver is needed

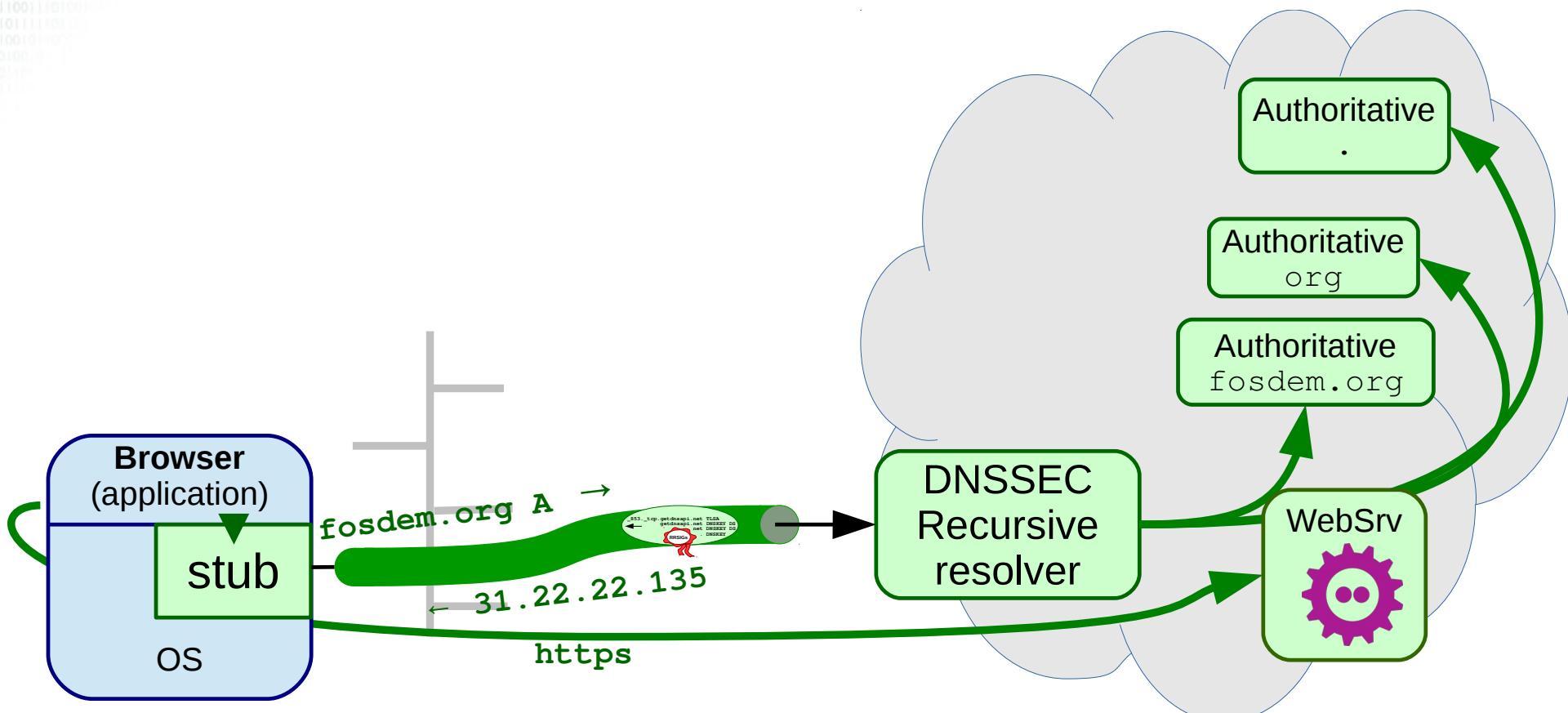
Encryption everywhere

2nd hurdle: DNSSEC Roadblocks



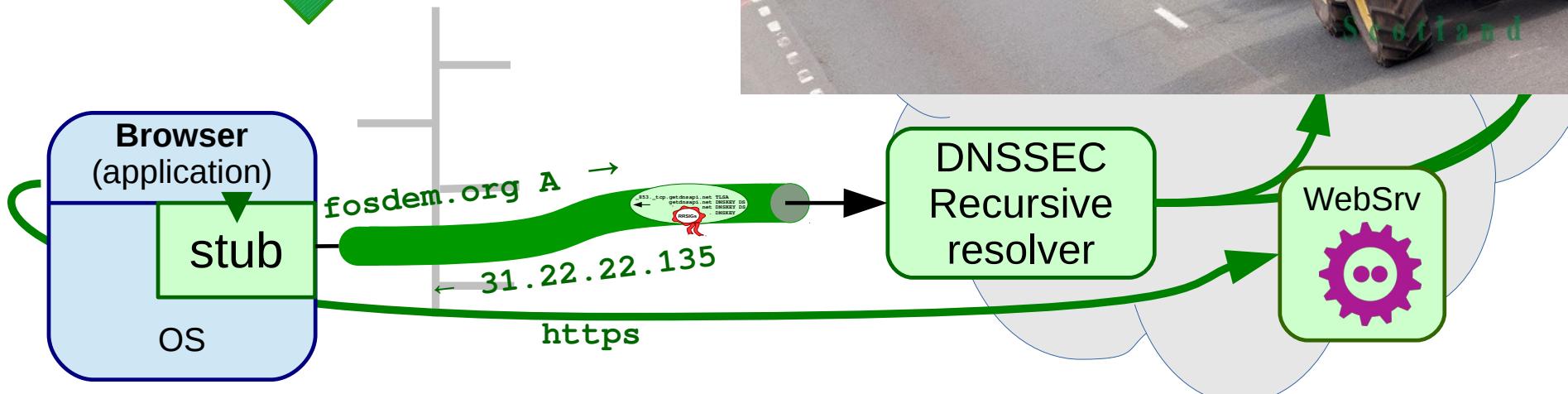
- A DNSSEC Aware recursive resolver is needed
- DNSSEC Roadblock Avoidance <https://tools.ietf.org/html/rfc8027> & Full recursion

Encryption everywhere



- DNS over TLS: RFC7858

Encryption

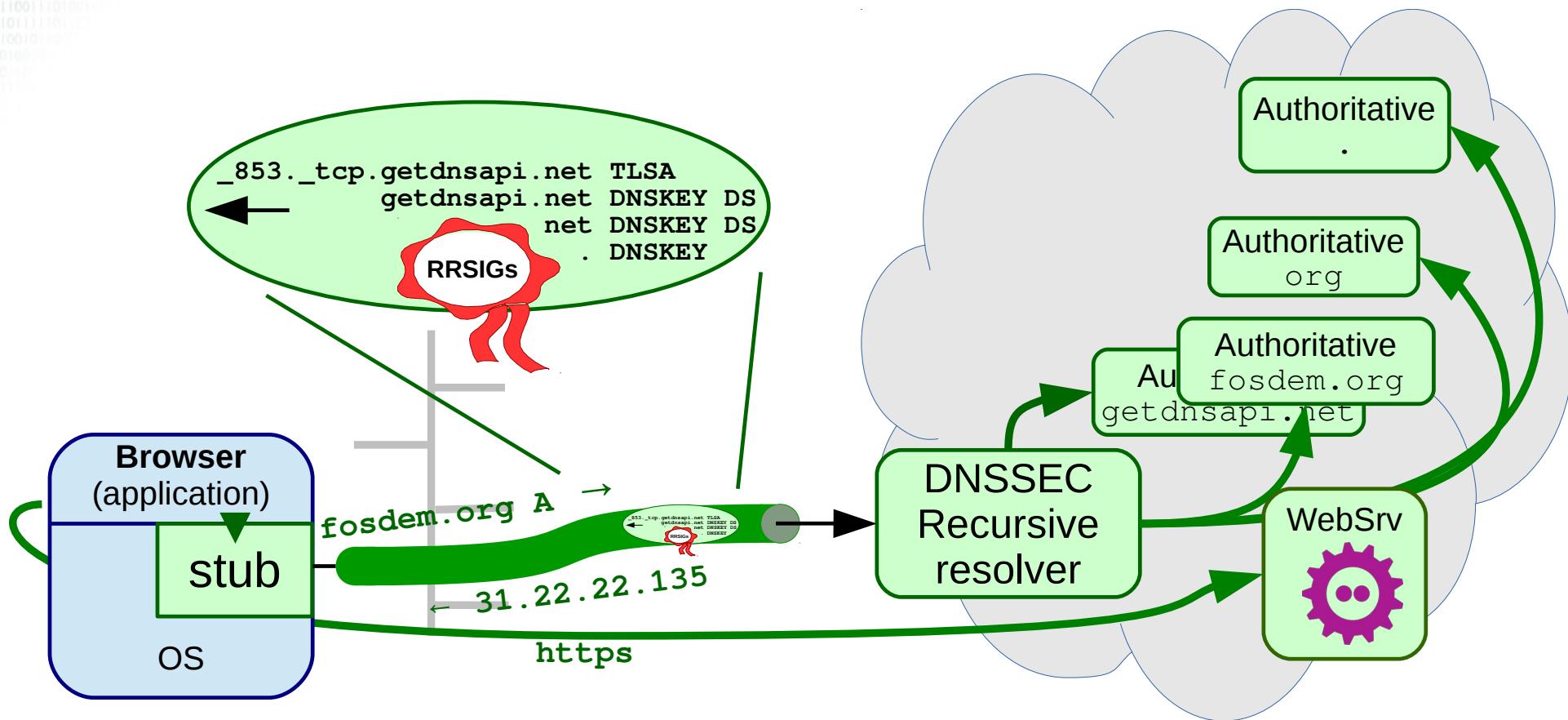


- DNS over TLS: RFC7858

Reuse / Pipelining / OOOR	RFC7766
TCP Fastopen	RFC7413
EDNS0 keepalive	RFC7828
EDNS0 padding	RFC7830

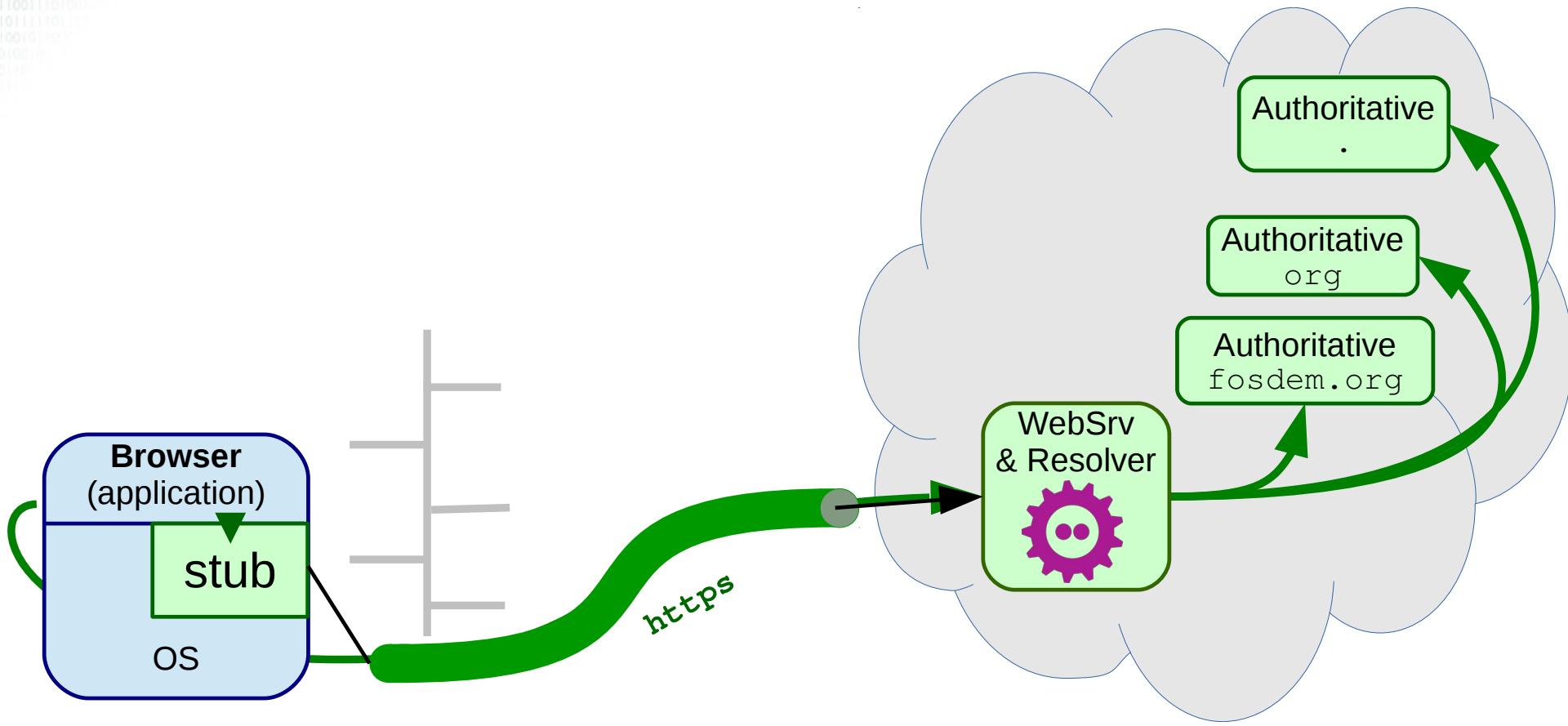


Encryption everywhere



- DNS over TLS: RFC7858
- Authenticating DNS over TLS
- [draft-ietf-tls-dnssec-chain-extension](#)
- [draft-bortzmeyer-dprive-resolver-to-auth](#)

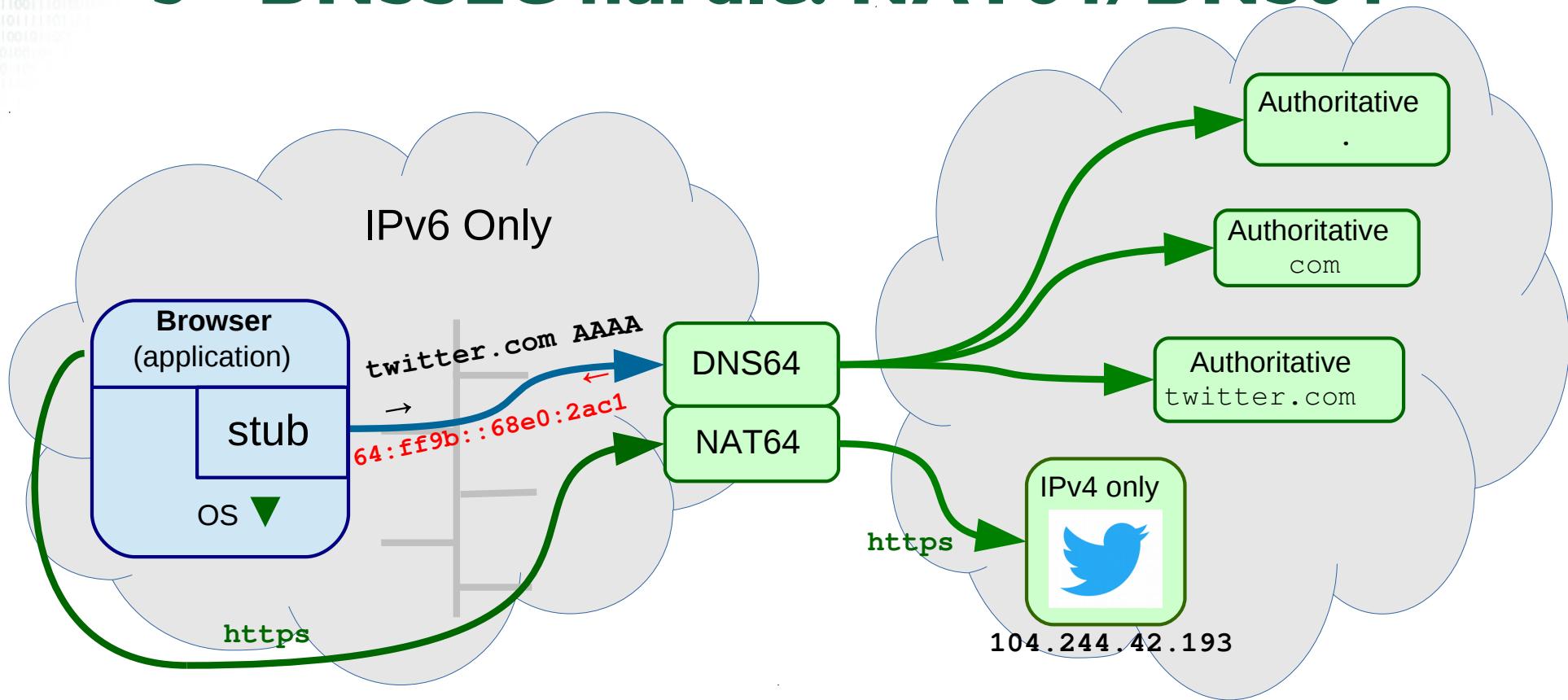
Encryption everywhere



- DNS over HTTPS:
[draft-ietf-doh-dns-over-https](https://datatracker.ietf.org/doc/draft-ietf-doh-dns-over-https)
- Queries in channel (less leakage)
- Port 443 less often blocked than port 853

Encryption everywhere

3rd DNSSEC hurdle: NAT64/DNS64

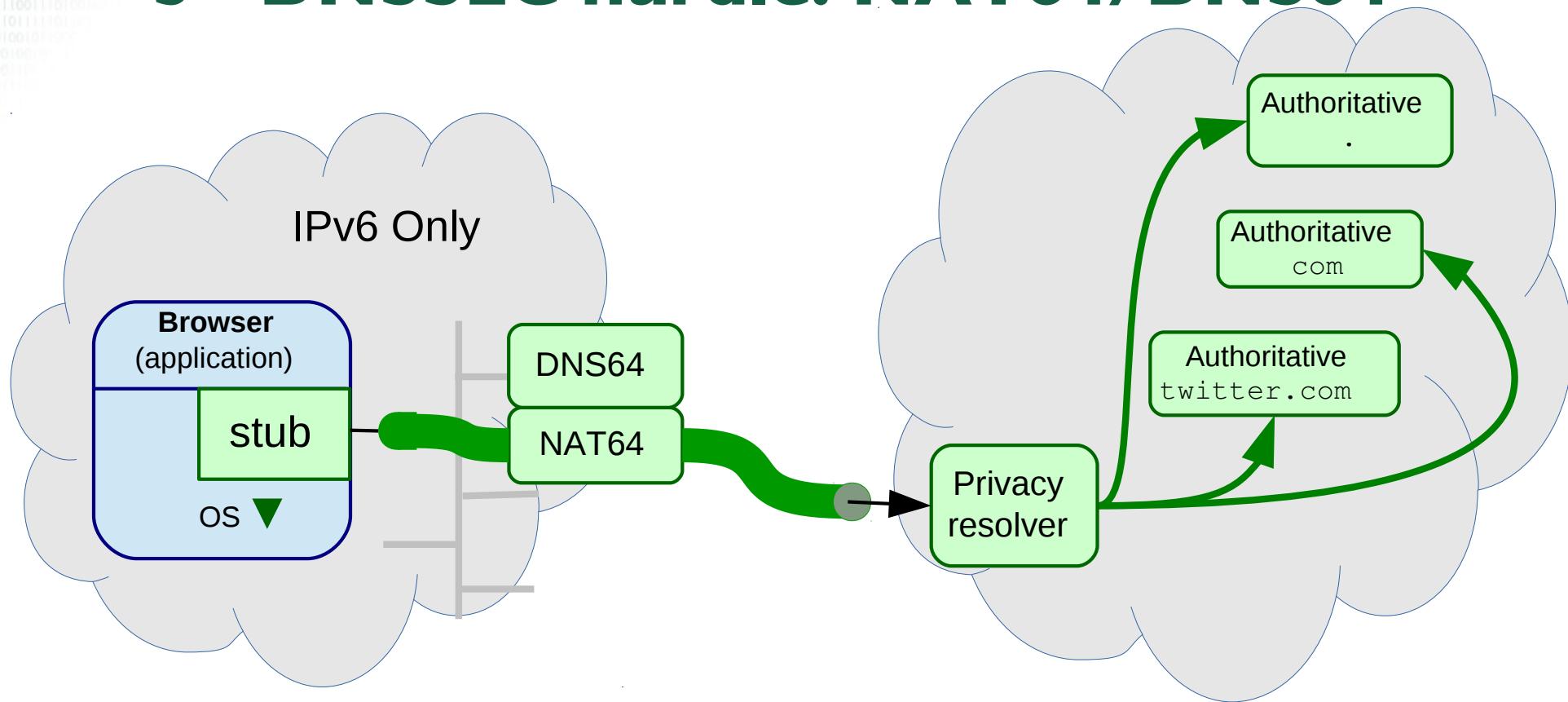


- DNSSEC Roadblock Avoidance <https://tools.ietf.org/html/rfc8027>
- IPv6 Address Synthesis Prefix Discovery
 - +DNS64 capability

<https://tools.ietf.org/html/rfc7050>
<https://tools.ietf.org/html/rfc6147>

Encryption everywhere

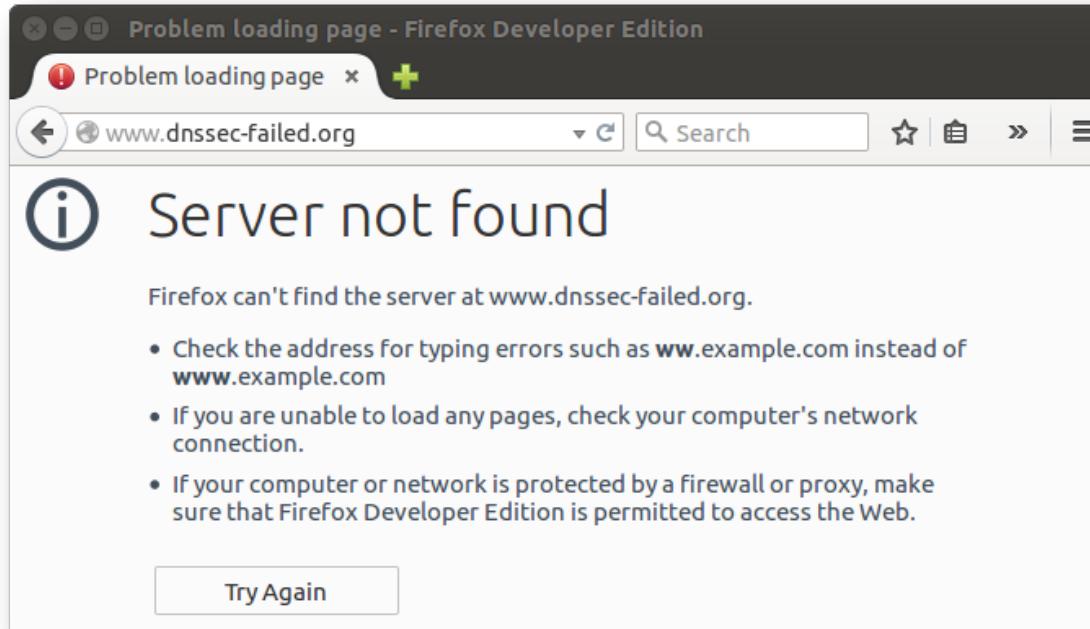
3rd DNSSEC hurdle: NAT64/DNS64



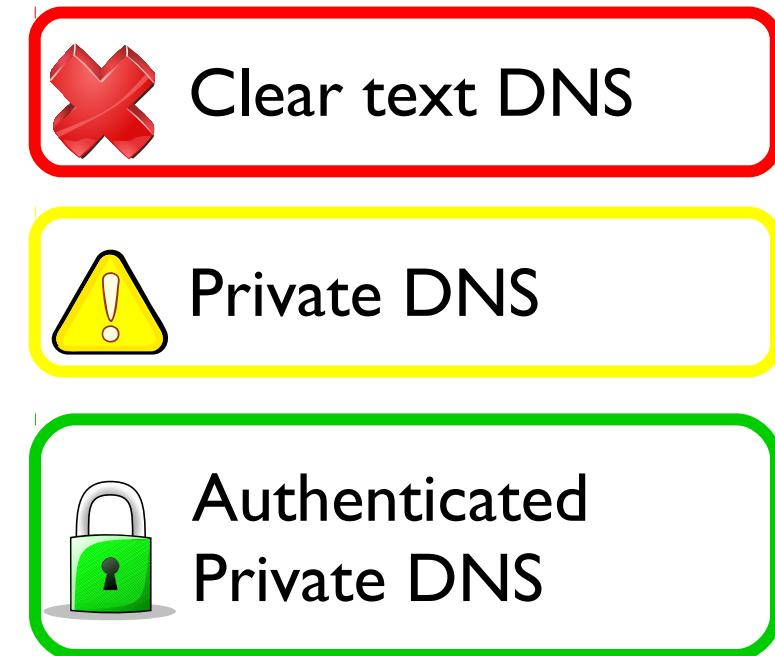
- DNSSEC Roadblock Avoidance <https://tools.ietf.org/html/rfc8027>
- IPv6 Address Synthesis Prefix Discovery
 - +DNS64 capability<https://tools.ietf.org/html/rfc7050>
<https://tools.ietf.org/html/rfc6147>

Yet another motivation for DNS sec/priv on the edge

DNSSEC Availability

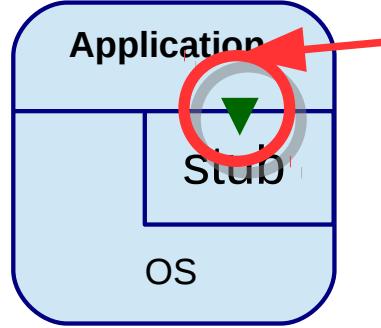


DNS Privacy status



- The stub is close to the application
- Inform status of DNSSEC and DNS Privacy

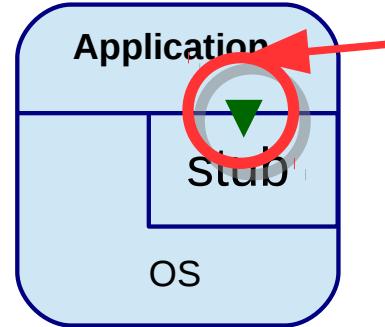
Application Interface



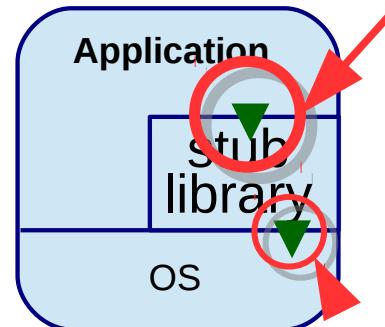
`getaddrinfo()` and `getnameinfo()`
(*POSIX standard extended by RFC3493 for IPv6*)

Application Interface

- non address lookups



getaddrinfo() and getnameinfo()
(POSIX standard extended by RFC3493 for IPv6)



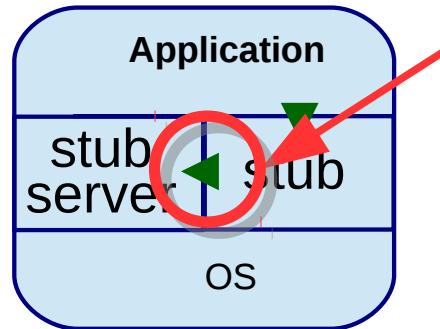
Talk to upstreams directly with a library:

- ~~libresolv~~, libval, ldns, libunbound, **libgetdns**

Learn upstreams from OS

- /etc/resolv.conf, NetworkManager, registry...

Application Interface - versatile stub server



Stub server listening on 127.0.0.1:53

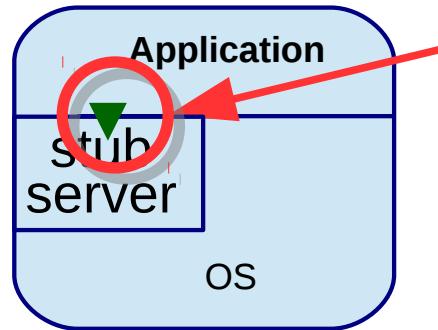
- `getaddrinfo()` and `getnameinfo()` use system stub which uses stub server



Dnsmasq

Application Interface

- versatile stub server

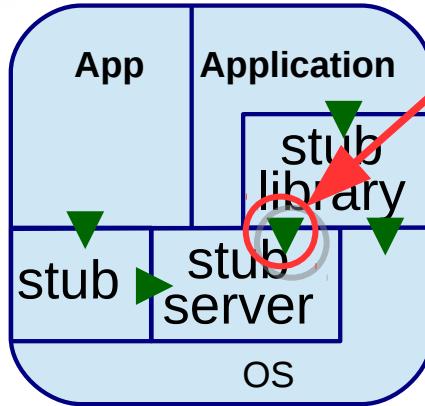


getaddrinfo() and getnameinfo()
use **systemd-resolved** via nsswitch module

- Stub server listening on 127.0.0.53:53

systemd-resolved.service
systemd-resolved

Application Interface - versatile stub server



Talk to stub server via a library:

- ~~libresolv~~, libval, ldns, libunbound, libgetdns



systemd-resolved.service
~~systemd-resolved~~
127.0.0.53:53

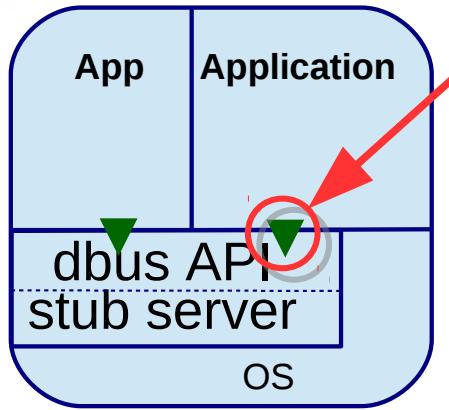


Dnssec-trigger



DnsMasq

Application Interface - versatile stub server

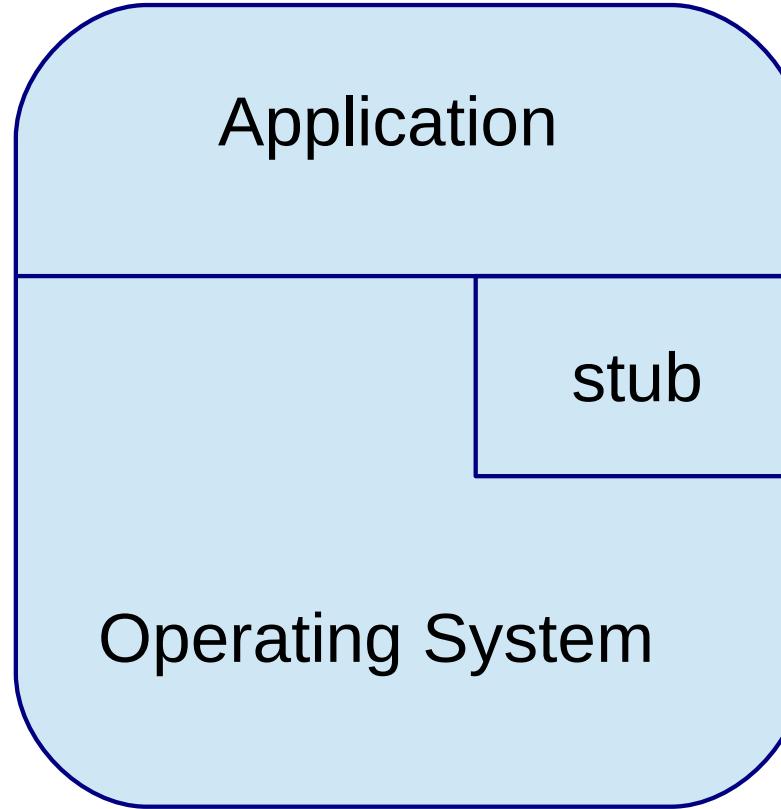


Talk to stub server via the dbus API

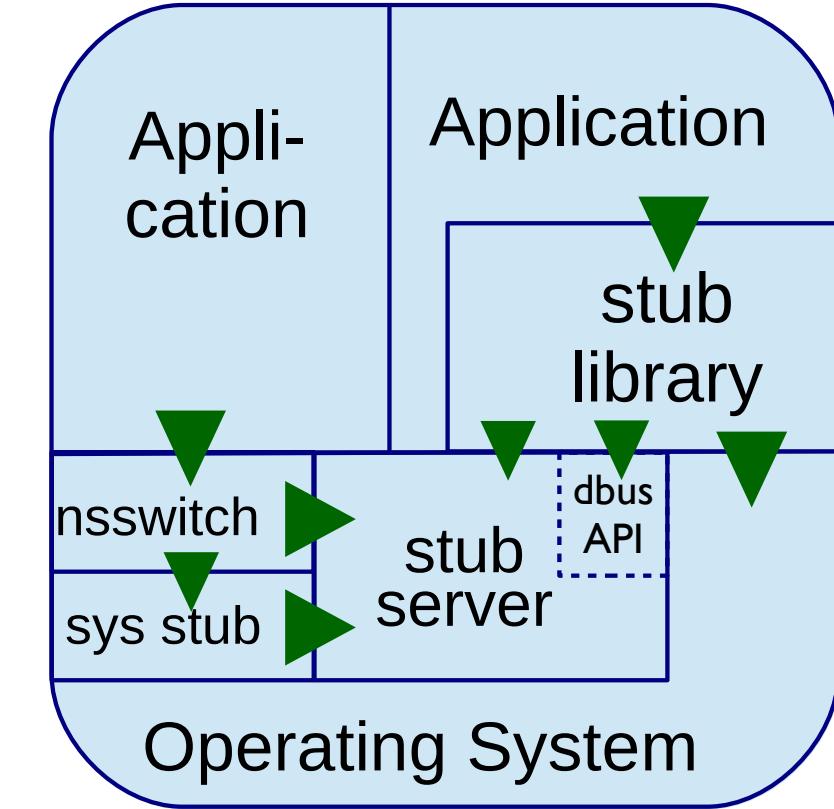
- <https://www.freedesktop.org/wiki/Software/systemd/resolved/>

systemd-resolved service
systemd resolved

Living on the Edge Encryption Everywhere



Before



After

GETDNS RELIEVES PROGRAMMERS FROM THE TASK OF HAVING TO DESIGN AND IMPLEMENT SECURE PARSING AND VALIDATION OF DOMAIN INFO THEMSELVES. INSTEAD, IT'S JUST A CALL AWAY



Hands on getdns slide deck after this slide...

Hands on *getdns*

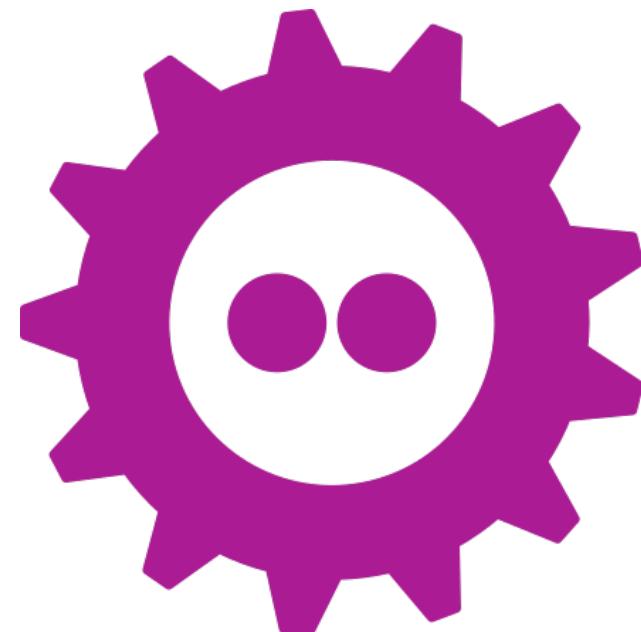
Unbound security

Sara Dickinson

sinodun

Willem Toorop

NLnet
Labs



FOSDEM'18

Hands on *getdns* Overview

- What is the getdns API
- Guided tour of the API
- Examples uses (code!)

but first...

getdns → installation on ubuntu

```
willem@bonobo: ~/afnic                               willem@bonobo: ~/afnic 80x24
willem@bonobo:~/afnic$ apt search getdns
Sorting... Done
Full Text Search... Done
getdns-utils/zesty 1.1.0~a2-1 amd64
  modern asynchronous DNS API (utils)

libgetdns-dev/zesty 1.1.0~a2-1 amd64
  modern asynchronous DNS API (development)

libgetdns1/zesty 1.1.0~a2-1 amd64
  modern asynchronous DNS API (shared library)

python-getdns/zesty 1.0.0~b1-1 amd64
  modern asynchronous DNS API (python bindings)

python-getdns-doc/zesty,zesty 1.0.0~b1-1 all
  modern asynchronous DNS API (documentation)

python3-getdns/zesty 1.0.0~b1-1 amd64
  modern asynchronous DNS API (python 3 bindings)

willem@bonobo:~/afnic$ sudo apt install getdns-utils python-getdns libgetdns-dev
```



installation on MacOS

```
willem$ brew info getdns
[getdns: stable 1.1.1 (bottled), HEAD
Modern asynchronous DNS API
https://getdnsapi.net
/usr/local/Cellar/getdns/1.1.1 (99 files, 1.7MB) *
  Poured from bottle on 2017-07-04 at 09:58:55
From: https://github.com/Homebrew/homebrew-core/blob/master/Formula/getdns.rb
==> Dependencies
Required: openssl ✓
Recommended: unbound ✓, libidn ✓, libevent ✓
Optional: libuv ✘, libev ✘
==> Options
--with-libev
    Build with libev support
--with-libuv
    Build with libuv support
--without-libevent
    Build without libevent support
--without-libidn
    Build without libidn support
--without-unbound
    Build without unbound support
--HEAD
    Install HEAD version
ieniemienie:~ willem$ brew install getdns
```



installation from tarball

```
$ wget https://getdnsapi.net/dist/getdns-1.3.0.tar.gz
```

```
--2017-07-04 10:20:20-- https://getdnsapi.net/dist/getdns-1.3.0.tar.gz
Resolving getdnsapi.net (getdnsapi.net)... 2a04:b900:0:100::37, 185.49.141.37
Connecting to getdnsapi.net (getdnsapi.net)|2a04:b900:0:100::37|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 948941 (927K) [application/x-gzip]
Saving to: 'getdns-1.3.0.tar.gz'

Getdns-1.3.0.tar.gz      100%[=====] 926.70K  ----KB/s   in 0.08s
2017-07-04 10:20:20 (11.9 MB/s) - 'getdns-1.3.0.tar.gz' saved [948941/948941]
```

```
$ tar xzf getdns-1.3.0.tar.gz
```

```
$ cd getdns-1.3.0/
```

```
$ ./configure --enable-stub-only --without-libidn
```

```
checking for gcc... gcc
```

```
checking whether the C compiler works... yes
```

```
checking for C compiler default output file name... a.out
```

```
checking for suffix of executables...
```

```
checking whether we are cross compiling... no
```

```
$ make
```

```
$ sudo make install
```



installation from repository

```
$ git clone https://github.com/getdnsapi/getdns
```

```
Cloning into 'getdns'...
remote: Counting objects: 13781, done.
remote: Compressing objects: 100% (165/165), done.
remote: Total 13781 (delta 167), reused 158 (delta 85), pack-reused 13531
Receiving objects: 100% (13781/13781), 8.86 MiB | 7.94 MiB/s, done.
Resolving deltas: 100% (10541/10541), done.
```

```
$ cd getdns
```

```
$ git checkout features/zeroconf-dnssec
```

```
Branch features/zeroconf-dnssec set up to track remote branch features/zeroconf-
dnssec from origin.
```

```
Switched to a new branch 'features/zeroconf-dnssec'
```

```
$ git submodule update --init
```

```
Submodule 'src/test/jsmn' (https://github.com/getdnsapi/jsmn.git) registered for
path 'src/jsmn'
Submodule 'src/yxml' (git://g.blicky.net/yxml.git) registered for path 'src/yxml'
Cloning into '/home/willem/getdns/getdns/src/jsmn'...
Cloning into '/home/willem/getdns/getdns/src/yxml'...
Submodule path 'src/jsmn': checked out '868c22e35ec223fc26ddefdb9ca83901dc6e2534'
Submodule path 'src/yxml': checked out '10f968b0e78b9aeee357d0de81a46b445c3fb27b'
```



installation from repository

```
$ autoreconf -fi
```

```
libtoolize: putting auxiliary files in '.'.  
libtoolize: copying file './ltmain.sh'  
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'm4'.  
libtoolize: copying file 'm4/libtool.m4'  
libtoolize: copying file 'm4/ltoptions.m4'  
libtoolize: copying file 'm4/ltsugar.m4'  
libtoolize: copying file 'm4/ltversion.m4'  
libtoolize: copying file 'm4/lt~obsolete.m4'  
libtoolize: Consider adding '-I m4' to ACLOCAL_AMFLAGS in Makefile.am.
```

```
$ glibtoolize -ci
```

```
libtoolize: putting auxiliary files in '.'.  
libtoolize: copying file './config.guess'  
libtoolize: copying file './config.sub'  
libtoolize: copying file './install-sh'  
libtoolize: Consider adding '-I m4' to ACLOCAL_AMFLAGS in Makefile.am.
```

```
$ ./configure --enable-stub-only --without-libidn
```

```
$ make
```

```
$ sudo make install
```



installation

try out getdns_query

```
$ getdns_query -h
```

```
usage: getdns_query [<option> ...] \
[@<upstream> ...] [+<extension> ...] ['{ <settings> }'] [<name>] [<type>]
```

```
default mode: recursive, synchronous resolution of NS record
using UDP with TCP fallback
```

```
upstreams: @<ip>[%<scope_id>][@<port>][#<tls port>][~<tls name>][^<tsig spec>]
           <ip>@<port> may be given as <IPv4>:<port>
           or '['<IPv6>[%<scope_id>]'']:<port> too
```

```
tsig spec: [<algorithm>:]<name>:<secret in Base64>
```

```
extensions:
```

```
+add_warning_for_bad_dns
+dnssec_return_status
+dnssec_return_only_secure
+dnssec_return_all_statuses
+dnssec_return_validation_chain
+dnssec_return_full_validation_chain
+dnssec_roadblock_avoidance
```

```
+edns_cookies
+return_both_v4_and_v6
+return_call_reporting
+sit=<cookie>          Send along cookie OPT
                           with value <cookie>
+specify_class=<class>
+0                      Clear all extensions
```



installation

try out getdns_query

```
$ getdns_query -i  
  
$ getdns_query -k  
$ getdns_query -s . DNSKEY +dnssec_return_status  
$ getdns_query -k  
$ ls -l $HOME/.getdns  
  
$ getdns_query -s @185.49.141.37~getdnsapi.net \  
    -l LTU +return_call_reporting  
  
$ getdns_query -s _443._tcp.www.afnic.fr TLSA \  
    +dnssec_return_validation_chain
```

The *getdns* API is:

- A *DNS API* specification
by and for application developers

(for resolving)

(for application)

motivation

- `getaddrinfo()` does not fit standards* any more
 - Protocol signalling in non-address records:
SSHFP, TLSA, OPENPGPKEY, SMIMEA,
URI, CAA, HIP, CDS, CDNSKEY, CSYNC, etc.
 - Asynchronous standards (Happy Eyeballs)
 - App. level DNSSEC validation (for DANE)
 - DNS Privacy

The API is:

- A *DNS API* specification
by and for application developers
 - (for resolving)
 - (for application)

From API design considerations:

motivation

- ... *There are other DNS APIs available, but there has been very little uptake ...*
- ... *talking to application developers*
- ... *the APIs were developed by and for DNS people, not application developers ...*
- ... *create a natural follow-on to getaddrinfo () ...*

The *getdns* API is:

- A *DNS API* specification
by and for application developers
 - (for resolving)
 - (for application)
- First edited by Paul Hoffman
- First published in April 2013
- Currently published at
<https://getdnsapi.net/documentation/spec/>
- Maintained by *the getdns team*

The *getdns* library is:

- An implementation of the *getdns* API
- A *DNS API* specification
by and for application developers
 - (for resolving)
 - (for application)
- First implementation initiative by Allison Mankin



VERISIGN™ LABS,

Melinda Shore and sinodun

NLnet
Labs,
1000011000
1101101000
1001101010
01100001100
00111100010
00010110100
00010110101

getdns
Unbound security

The *getdns* library is:

- 26 February 2014: getdns-0.1.0 release
- 23 July 2015: took over editor role of the API specification
- 22 October 2015: New API specification release introducing JSON-pointers
- 2016: 2 getdns-1.0.0 beta releases
2 getdns-1.1.0 alpha releases
- 17 January 2017: getdns-1.0.0 released
100% specification complete
- 3 July 2017: getdns-1.1.2 released (latest)
many non-API functions

non-API doc: <https://getdnsapi.net/doxygen/modules.html>

The *getdns* library is:

- An implementation of the *getdns* API

Claus Assman, Theogene Bucuti, Andrew Cathrow, Neil Cook, Saúl Ibarra Corretgé, Craig Despeaux, John Dickinson, **Sara Dickinson**, Robert Edmonds, Angelique Finan, Simson Garfinkel, Daniel Kahn Gillmor, Neel Goyal, Bryan Graham, Robert Groenengen, Jim Hague, Paul Hoffman, Scott Hollenbeck, **Christian Huitema**, **Shumon Huque**, Jelte Janssen, Guillem Jover, Shane Kerr, Anthony Kirby, Olaf Kolkman, Sanjay Mahurpawar, **Allison Mankin**, Sai Mogali, Linus Nordberg, **Benno Overeinder**, **Joel Purra**, Tom Pusateri, Prithvi Ranganath, **Hoda Rohani**, Rushi Shah, Vinay Soni, **Melinda Shore**, Bob Steagall, Andrew Sullivan, Ondřej Surý, **Willem Toorop**, Gowri Visweswaran, **Wouter Wijngaards**, Glen Wiley, Paul Wouters

- Weekly meetings with the *getdns core team*

The *getdns* library

- Core team active in IETF and at IETF hackathons:
 - “Best in Show” prize at IETF93
DNSSEC roadblock detection, start of DNS over TLS
 - “Best internet security” at IETF94
edns0-client-subnet privacy election, start of padding
 - IETF95 – start of TLS DNSSEC auth. chain ext.
 - IETF96 – start of DNS64 work
 - IETF97 – Stubby interoperability testing
 - IETF98 – Start of Zero Configuration DNSSEC and...

The library motivation

From the README.md:

... DNSSEC offers a unique global infrastructure for establishing cryptographic trust relations ...

... offer application developers a modern and flexible way that enables end-to-end trust in the DNS architecture ...

... inspire application developers towards innovative security solutions ...



related research and projects

- The role of a versatile stub in *from the ground up* privacy (and security)
- DNSSEC for legacy applications
(i.e. a dnssec-trigger follow up, with signalling)
- How can getdns benefit from a system component
 - i.e. share stateful connections to upstreams
 - have feedback on level of privacy

getdns

related research
and projects



Stubby



look & feel data structures

- *getdns* is not your typical C library

```
typedef struct getdns_dict getdns_dict;
typedef struct getdns_list getdns_list;
typedef struct getdns_bindata { size_t size;
                                uint8_t *data; } getdns_bindata;
```

- Script like data structures are used to represent:
 - DNS responses
 - Resource Records
 - Rdata fields
- Formatted as JSON-like strings by

```
char *getdns_pretty_print_dict(getdns_dict *dict);
char *getdns_pretty_print_list(getdns_list *list);
```



look & feel data structures

- *getdns* is not your typical C library

response dict

```
{ "answer_type": GETDNS_NAMETYPE_DNS,  
  "status": GETDNS_RESPSTATUS_GOOD  
  "canonical_name": <bindata for afnic.fr.>,  
  "just_address_answers": [  
    { "address_data": <bindata for 192.134.5.25>,  
      "address_type": <bindata of "IPv4"> },  
    { "address_data": <bindata for 2001:67c:2218:30::5>,  
      "address_type": <bindata of "IPv6"> }  
  ],  
  "replies_full": [  
    <bindata of 0x7a2f81b000010002000400010561666e...>,  
    <bindata of 0xa40581b000010002000400010561666e...>  
  ],  
  "replies_tree": [{ ... first reply ... }, { ... second reply ... }]  
}
```



look & feel data structures

- *getdns* is not your typical C library
reply

```
{ "answer_type": GETDNS_NAMETYPE_DNS,  
  "canonical_name": <bindata for afnic.fr.>,  
  "dnssec_status": GETDNS_DNSSEC_SECURE,  
  "header": { "id": 50407,  
             "qr": 1, "opcode": GETDNS_OPCODE_QUERY,  
             "aa": 0, "tc": 0, "rd": 1, "ra": 1,  
             "z": 0, "ad": 1, "cd": 1,  
             "rcode": GETDNS_RCODE_NOERROR,  
             "qdcnt": 1, "ancount": 2,  
             "nscount": 4, "arcount": 1  
           },  
  "question": { "qclass": GETDNS_RRCLASS_IN,  
               "qname": <bindata for afnic.fr.>,  
               "qtype": GETDNS_RRTYPE_A  
             },
```



look & feel data structures

- *getdns* is not your typical C library
reply

```
"answer": [  
    { "name" : <bindata for afnic.fr.>,  
     "type" : GETDNS_RRTYPE_A,  
     "class": GETDNS_RRCLASS_IN,  
     "ttl"  : 31,  
     "rdata": { "ipv4_address": <bindata for 192.134.5.25>,  
                "rdata_raw": <bindata of 0xc0860519>  
               }  
    },  
    { "name" : <bindata for afnic.fr.>,  
     "type" : GETDNS_RRTYPE_RRSIG,  
     "rdata": { "type_covered": GETDNS_RRTYPE_A,  
                "algorithm": 8, "labels": 2,  
                "original_ttl": 600,  
                "signature_expiration": 1500960505,  
                "signature_inception": 1498356836,  
                "key_tag": 16774,  
                "signers_name": <bindata for afnic.fr.>,  
               }  
    }]
```

Do a Query - Mozilla Firefox

Do a Query | https://getdnsapi.net/query?hostname=afi | Zoeken

getdns

Quick Start Documentation Presentations Releases

Do a Query

afnic.fr A Query »

Extensions

return_both_v4_and_v6 dnssec_return_status
 return_call_reporting dnssec_return_only_secure
 add_warning_for_bad_dns dnssec_return_validation_chain
 dns64 dnssec_return_all_statuses

Transport

Transport order: UDP, TCP
TLS resolver IP: 185.49.141.38
TLS auth name: getdnsapi.net

SYNC response:

```
{  
  "answer_type": GETDNS_NAMETYPE_DNS,  
  "canonical_name": <bindata for afnic.fr.>,  
  "just_address_answers":  
  [  
    {  
      "address_data": <bindata for 192.134.5.25>,  
      "label": "192.134.5.25",  
      "type": "A"  
    }  
  ]  
}
```



look & feel data structures

- reading getdns_dicts:

```
getdns_return_t getdns_dict_get_dict(  
    const getdns_dict *dict, const char *name, getdns_dict **answer);  
  
getdns_return_t getdns_dict_get_list(  
    const getdns_dict *dict, const char *name, getdns_list **answer);  
  
getdns_return_t getdns_dict_get_bindata(  
    const getdns_dict *dict, const char *name, getdns_bindata **answer);  
  
getdns_return_t getdns_dict_get_int(  
    const getdns_dict *dict, const char *name, uint32_t *answer)  
  
getdns_return_t getdns_dict_get_data_type(  
    const getdns_dict *dict, const char *name, getdns_data_type *answer);  
  
getdns_return_t getdns_dict_get_names(  
    const getdns_dict *dict, getdns_list **answer);
```



look & feel data structures

- reading getdns_lists:

```
getdns_return_t getdns_list_get_dict(  
    const getdns_list *list, size_t index, getdns_dict **answer);  
  
getdns_return_t getdns_list_get_list(  
    const getdns_list *list, size_t index, getdns_list **answer);  
  
getdns_return_t getdns_list_get_bindata(  
    const getdns_list *list, size_t index, getdns_bindata **answer);  
  
getdns_return_t getdns_list_get_int(  
    const getdns_list *list, size_t index, uint32_t *answer);  
  
getdns_return_t getdns_list_get_data_type(  
    const getdns_list *list, size_t index, getdns_data_type *answer);  
  
getdns_return_t getdns_list_get_length(  
    const getdns_list *this_list, size_t *answer);
```



look & feel data structures

- Creating/writing to getdns_dicts:

```
getdns_dict * getdns_dict_create();  
  
getdns_return_t getdns_dict_set_dict(  
    getdns_dict *dict, const char *name, const getdns_dict *child_dict);  
  
getdns_return_t getdns_dict_set_list(  
    getdns_dict *dict, const char *name, const getdns_list *child_list);  
  
getdns_return_t getdns_dict_set_bindata(  
    getdns_dict *dict, const char *name, const getdns_bindata  
*child_bindata);  
  
getdns_return_t getdns_dict_set_int(  
    getdns_dict *dict, const char *name, uint32_t child_uint32)  
  
void getdns_dict_destroy(getdns_dict *dict);
```

getdns look & feel data structures

Response object dict

```
{  
    "answer_type": GETDNS_NAMETYPE_DNS,  
    "status": GETDNS_RESPSTATUS_GOOD,  
    "canonical_name": <bindata of "www.getdnsapi.net.">,  
    "just_address_answers":  
        [ { "address_data": <bindata for 185.49.141.37>,  
            "address_type": <bindata of "IPv4">  
        }  
        ],  
    "replies_full": [ <bindata of 0x00008180000100020004...> ],  
    "replies_tree": [ { ... first reply ... } ],
```

```
if ((r = getdns_address_sync(ctx, "getdnsapi.net", ext, &resp)))  
    return r;  
else if ((r = getdns_list_get_bindata(  
    addr_dict, "/just_address_answers/0/address_data", &addr)))  
    return r;
```



look & feel data structures

```
if ((r = getdns_address_sync(ctx, "getdnsapi.net", ext, &resp)))
    return r;
else if ((r = getdns_list_get_bindata(
    addr_dict, "/just_address_answers/0/address_data", &addr)))
    return r;
```

- Natural in script languages

- Python

```
resp = ctx.address('getdnsapi.net')
addr = resp.just_address_answers[0]['address_data']
```

- Nodejs

```
function callback(err, resp) {
    var addr = resp.just_address_answers[0].address_data;
}
ctx.getAddress('getdnsapi.net', callback);
```



look & feel data structures

- The alternative would introduce a lot of new types:

- Python:

```
addr = resp.replies_tree[0]['answer'][0]['rdata']['ipv6_address']
```

- C now

```
r = getdns_dict_get_bindata(  
    resp, "/replies_tree/0/answer/rdata/ipv6_address", &addr))
```

- C otherwise (ldns like)

```
getdns_response *resp; getdns_reply     *reply;  
getdns_rrs      *rrs;  getdns_rr       *rr;  
getdns_rdata    *rdata; struct sockaddr_storage addr;  
if ((r = getdns_response_get_reply(resp, 0, &reply)))  
    return r;  
else if ((r = getdns_reply_get_answer_section(reply, &rrs)))  
    return r;  
else if ((r = getdns_rrs_get_rr(rrs, &rr)))  
    return r;  
else if ((r = getdns_rr_get_rdata(rr, &rdata)))  
    return r;  
else if ((r = getdns_rdata_get_rdatafield_address(rdata, 0, &addr)))  
    return r;
```



look & feel data structures

- *getdns* is not your typical C library
- Natural in script languages
- The alternative would introduce a lot of new types.
- With current approach, the library can easily grow
- New rdata fields or new extensions without a new API (*dns cookies, roadblock avoidance, client subnet, etc.*)
- Just in time parsing of wireformat data on the roadmap (*internally already iterator like accessor types for wireformat data*)
- Still... “C *bindings*” on the roadmap



look & feel lookup functions

```
getdns_return_t getdns_general(  
    getdns_context *context,  
    const char *name,  
    uint16_t request_type,  
    getdns_dict *extensions,  
    void *userarg,  
    getdns_transaction_t *transaction_id,  
    getdns_callback_t callbackfn  
);
```

- **context** contains configuration parameters
 - Stub or recursive modus operandi, timeout values, root-hints, forwarders, trust anchor, search path etc.)
- **context** contains the resolver cache
- `getdns_return_t getdns_context_create(
 getdns_context **context, int set_from_os)`



look & feel lookup functions

```
getdns_return_t getdns_general(  
    getdns_context           *context,  
    const char               *name,  
    uint16_t                request_type,  
    getdns_dict              *extensions,  
    void                     *userarg,  
    getdns_transaction_t    *transaction_id,  
    getdns_callback_t        callbackfn  
);
```

- context contains configuration parameters
- **name** and **request_type** the name and type to lookup



look & feel lookup functions

```
getdns_return_t getdns_general(  
    getdns_context           *context,  
    const char                *name,  
    uint16_t                  request_type,  
    *extensions,  
    void                      *userarg,  
    getdns_transaction_t     *transaction_id,  
    getdns_callback_t         callbackfn  
);
```

- context contains configuration parameters
- name and request_type the name and type to lookup
- **extensions** additional parameters specific for this lookup
 - `return_both_v4_and_v6`, `dnssec_return_status`, `specify_class`, `add_opt_parameter`



look & feel extensions

- `dnssec_return_validation_chain`

```
- { # Response object
  "validation_chain":
  [ { "name" : <bindata for .>, "type": GETDNS_RRTYPE_DNSKEY, ... },
    { "name" : <bindata for .>, "type": GETDNS_RRTYPE_DNSKEY, ... },

    { "name" : <bindata for .>, "type": GETDNS_RRTYPE_RRSIG,
      "rdata": { "signers_name": <bindata for .>,
                 "type_covered": GETDNS_RRTYPE_DNSKEY, ... }, ... },
    { "name" : <bindata for net.>, "type": GETDNS_RRTYPE_DS, ... },
    { "name" : <bindata for net.>, "type": GETDNS_RRTYPE_RRSIG,
      "rdata": { "signers_name": <bindata for .>,
                 "type_covered": GETDNS_RRTYPE_DS, ... }, ... },
```

- Can be fed as `support_records` with companion function:

- `getdns_return_t`
`getdns_validate_dnssec(getdns_list *to_validate`
 `, getdns_list *support_records`
 `, getdns_list *trust_anchors);`



look & feel lookup functions

```
getdns_return_t getdns_general(  
    getdns_context           *context,  
    const char                *name,  
    uint16_t                  request_type,  
    getdns_dict               *extensions,  
    void                      *userarg,  
    getdns_transaction_t     *transaction_id,  
    getdns_callback_t         callbackfn  
);
```

- context contains configuration parameters
- name and request_type the name and type to lookup
- extensions additional parameters specific for this lookup
- *userarg* is passed in on the call to *callbackfn*
- *transaction_id* is set to a unique value that is also passed in on the call to *callbackfn*

getdns look & feel lookup functions

```
getdns_return_t getdns_general(  
    getdns_context           *context,  
    const char                *name,  
    uint16_t                  request_type,  
    getdns_dict               *extensions,  
    void                      *userarg,  
    getdns_transaction_t     *transaction_id,  
    getdns_callback_t         callbackfn  
);  
  
typedef void (*getdns_callback_t)(  
    getdns_context           *context,  
    getdns_callback_type_t   callback_type,  
    getdns_dict               *response,  
    void                      *userarg,  
    getdns_transaction_t     transaction_id  
);  
// callback_type = complete, cancel, timeout or error
```



look & feel lookup functions

```
getdns_return_t getdns_general(  
    getdns_context           *context,  
    const char                *name,  
    uint16_t                  request_type,  
    getdns_dict               *extensions,  
    void                      *userarg,  
    getdns_transaction_t     *transaction_id,  
    getdns_callback_t         callbackfn  
);  
  
getdns_return_t getdns_general_sync(  
    getdns_context           *context,  
    const char                *name,  
    uint16_t                  request_type,  
    getdns_dict               *extensions,  
    getdns_dict               **response  
);
```



look & feel lookup functions

```
getdns_return_t getdns_address(  
    getdns_context           *context,  
    const char                *name,  
    getdns_dict               *extensions,  
    void                      *userarg,  
    getdns_transaction_t     *transaction_id,  
    getdns_callback_t         callbackfn  
) ;
```

- **getdns_address** also lookups in other name systems
 - local files, mDNS (not implemented yet)
- When name is found in the DNS, **getdns_address** returns both IPv4 and IPv6



look & feel lookup functions

```
getdns_return_t getdns_hostname(  
    getdns_context *context,  
    getdns_dict *address,  
    getdns_dict *extensions,  
    void *userarg,  
    getdns_transaction_t *transaction_id,  
    getdns_callback_t callbackfn  
) ;
```

- With **address**: { "address_type": <bindata of "IPv4">,
"address_data": <bindata for 185.49.141.37>
}

will lookup 37.141.49.185.in-addr.arpa PTR



look & feel lookup functions

```
getdns_return_t getdns_service(  
    getdns_context           *context,  
    getdns_dict              *name,  
    getdns_dict               *extensions,  
    void                      *userarg,  
    getdns_transaction_t     *transaction_id,  
    getdns_callback_t         callbackfn  
) ;
```

- Provides a partly randomly sorted list (by weight and priority) of service addresses and ports (RFC2782)

```
"canonical_name": <bindata for  
"jabber._tcp.nlnetlabs.nl.">  
"srv_addresses":  
  [ { "address_data": <bindata for  
2a04:b900::1:0:0:10>,  
      "domain_name": <bindata for  
open.nlnetlabs.nl.>,  
      "port": 5222 } ]
```



look & feel event libraries

- **libevent**

Include : **#include <getdns/getdns_ext_libevent.h>**

Use: **getdns_extension_set_libevent_base(context, base);**

Link : -lgetdns -lgetdns_ext_event

- **libev**

Include : **#include <getdns/getdns_ext_libev.h>**

Use: **getdns_extension_set_libev_loop(context, loop);**

Link : -lgetdns -lgetdns_ext_ev

- **libuv**

Include : **#include <getdns/getdns_ext_libuv.h>**

Use: **getdns_extension_set_libuv_loop(context, loop);**

Link : -lgetdns -lgetdns_ext_uv

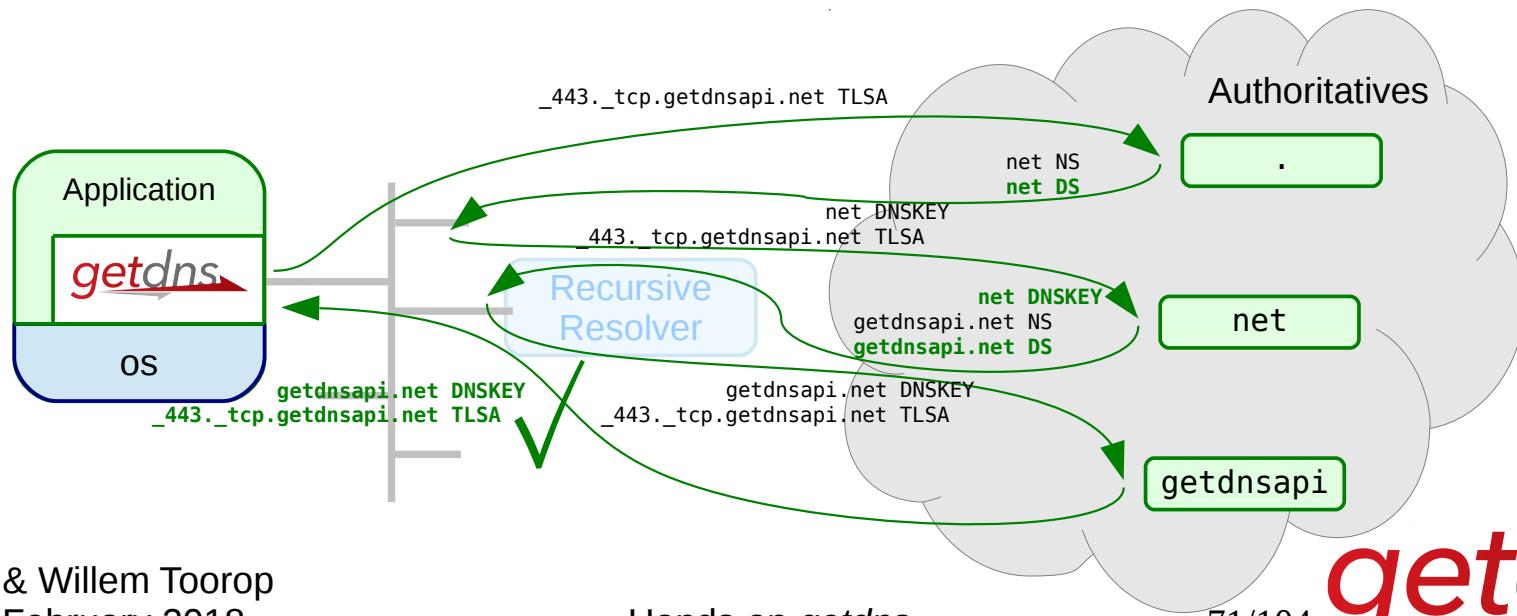


look & feel example query

```
from getdns import *

ctx = Context()
ext = { "dnssec_return_only_secure": EXTENSION_TRUE }
res = ctx.general( '_443._tcp.getdnsapi.net', RR_TYPE_TSLA, ext)

if res['status'] == RESPSTATUS_GOOD:
    # Process TSLA RRs
```





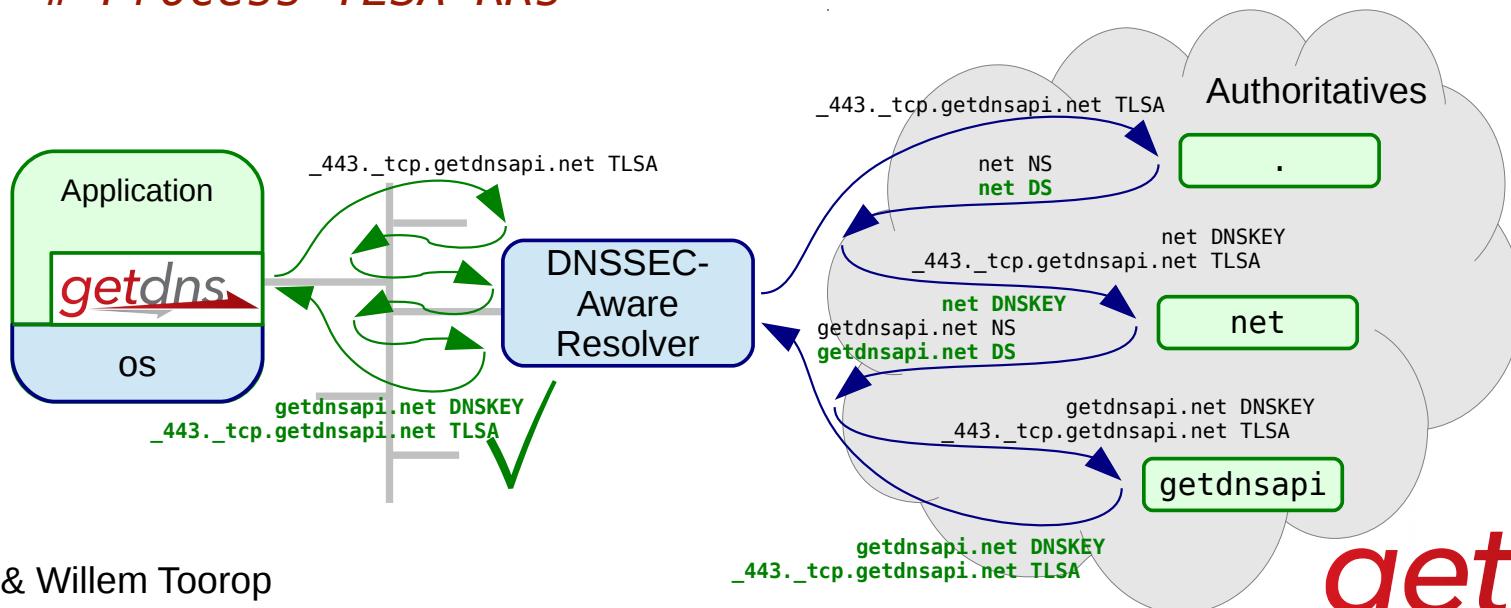
look & feel example query

```
from getdns import *

ctx = Context()
ctx.resolution_type = RESOLUTION_STUB

ext = { "dnssec_return_only_secure": EXTENSION_TRUE }
res = ctx.general( '_443._tcp.getdnsapi.net', RR_TYPE_TSLA, ext)

if res['status'] == RESPSTATUS_GOOD:
    # Process TSLA RRs
```





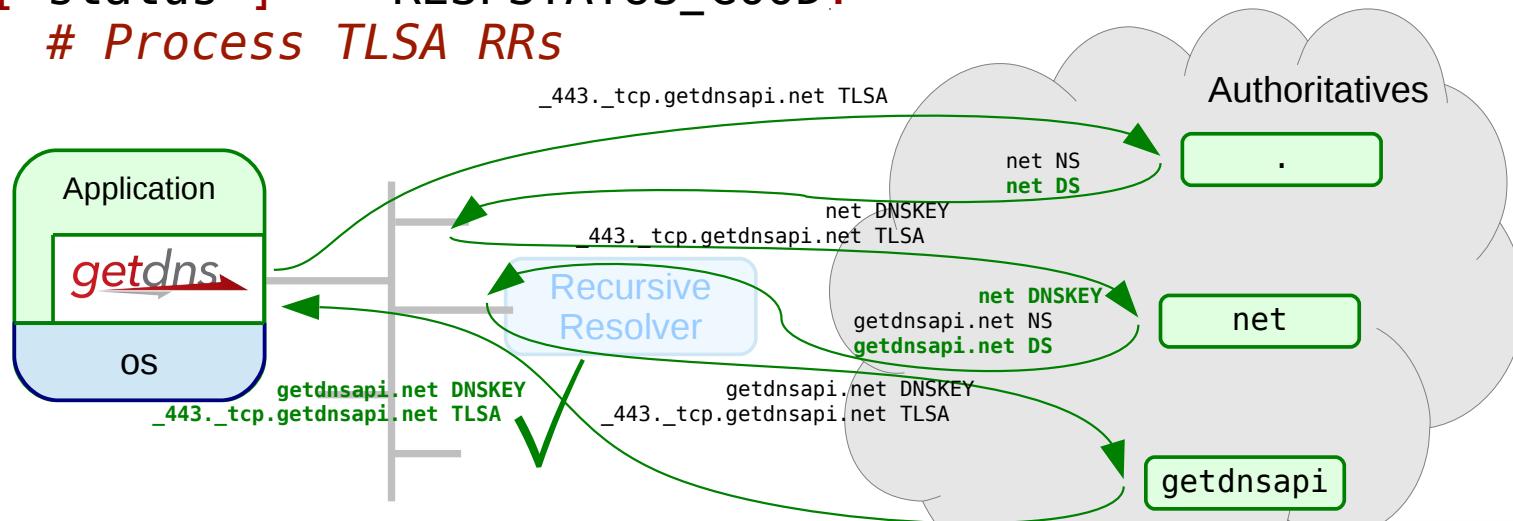
look & feel example query

```
from getdns import *

ctx = Context()
ctx.resolution_type = RESOLUTION_STUB

ext = { "dnssec_return_only_secure" : EXTENSION_TRUE
       , "dnssec_roadblock_avoidance": EXTENSION_TRUE}
res = ctx.general( '_443._tcp.getdnsapi.net', RR_TYPE_TSLA, ext)

if res['status'] == RESPSTATUS_GOOD:
    # Process TLSA RRs
```





look & feel example query

```
from getdns import *

def process_tlsa_rrs( ttype, result, userarg, tid ):
    ctx = userarg
    if ttype == CALLBACK_COMPLETE:
        # Process TLSA RRs
        pass
    elif ttype == CALLBACK_TIMEOUT:
        # Handle timeout
        pass

ctx = Context()
ctx.resolution_type = RESOLUTION_STUB

ext = { "dnssec_return_only_secure" : EXTENSION_TRUE
       , "dnssec_roadblock_avoidance": EXTENSION_TRUE}
tid = ctx.general( '_443._tcp.getdnsapi.net', RR_TYPE_TSLA, ext
                  , userarg = ctx, callback = process_tlsa_rrs )
ctx.run()
```



look & feel example query

```
var getdns = require('getdns');

function process_tlsa_rrs(err, res)
{
    if (err) {
        console.log( err )
    } else {
        // Process TLSA RRs
    }
}

ctx = getdns.createContext();
ctx.general( '_443._tcp.getdnsapi.net', getdns.RRTYPE_TLSA
            , { dnssec_return_only_secure : true
              , dnssec_roadblock_avoidance: true }
            , function(err, res) { process_tlsa_rrs(ctx, err, res); });
```

Hands on



DANE authenticated TLS connect (*python*)

```
from getdns import *
from M2Crypto import SSL, X509
import sys
from socket import *
import hashlib

if len(sys.argv) > 1:
    hostname = sys.argv[1]
    port = int(sys.argv[2]) if len(sys.argv) > 2 else 443
else:
    print('%s <hostname> [ <port> ]' % sys.argv[0])
    sys.exit(0)

ctx = Context()
ctx.resolution_type = RESOLUTION_STUB
ext = { "dnssec_return_only_secure" : EXTENSION_TRUE }
#      , "dnssec_roadblock_avoidance": EXTENSION_TRUE }

# Correctly query and process DANE records
res = ctx.general('_%d._tcp.%s' % (port, hostname), RR_TYPE_TSLA, ext)
```

Hands on



DANE authenticated TLS connect (*python*)

```
if res.status == RESPSTATUS_GOOD:  
    # Process TLSA Rrs  
    tlsas = [ answer for reply in res.replies_tree  
              for answer in reply['answer']  
              if answer['type'] == RR_TYPE_TLSA ]  
  
elif res.status == RESPSTATUS_ALL_TIMEOUT:  
    print('Network error trying to get DANE records for %s' % hostname)  
    sys.exit(-1);  
elif res.status == RESPSTATUS_ALL_BOGUS_ANSWERS:  
    print('DANE records for %s were BOGUS' % hostname)  
    sys.exit(-1);  
else:  
    tlsas = None  
    # Conventional PKIX without DANE processing
```

Hands on



DANE authenticated TLS connect (*python*)

- Find the CA vouching for the connection for PKIX-TA and DANE-TA usages.

```
ca_cert = None
def get_ca(ok, store):
    global ca_cert
    if store.get_current_cert().check_ca():
        ca_cert = store.get_current_cert()
    return ok
```

```
# Now TLS connect to each address and verify the cert (or CA)
for address in ctx.address(hostname).just_address_answers:
    sock = socket(AF_INET if address['address_type'] == 'IPv4'
                  else AF_INET6, SOCK_STREAM)
    socket.setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, 1)
    print('Connecting to %s' % address['address_data'])
    ssl_ctx = SSL.Context()
    ssl_ctx.load_verify_locations(capath = '/etc/ssl/certs')
    ssl_ctx.set_verify(SSL.verify_none, 10, get_ca)
    connection = SSL.Connection(ssl_ctx, sock=sock)
```

- Just two more household affairs...

```
# set TLS SNI extension
connection.set_tlsext_host_name(hostname)

# Per RFC7671, for DANE-EE usage, certificate identity checks are
# based solely on the TLSA record, so we ignore name mismatch
# conditions in the certificate.
try:
    connection.connect((address['address_data'], port))

except SSL.Checker.WrongHost:
    pass
```

Hands on



DANE authenticated TLS connect (*python*)

- Without TLSA RRs, fall back to old fashioned PKIX

```
if not tlsas:  
    print( 'No TLSAS. Regular PKIX validation '  
          + ('succeeded' if connection.verify_ok() else 'failed'))  
    continue # next address
```

- But with TLSA RRs, try each TLSA RR in turn.
First one matching makes the day!
- Note that for PKIX-TA (0) and DANE-TA (2) we set cert to the CA.

```
cert = connection.get_peer_cert()  
TLSA_matched = False  
for tlsa in tlsas:  
    rdata = tlsa['rdata']  
    if rdata['certificate_usage'] in (0, 2):  
        cert = ca_cert
```

- Put certdata into selector and the matching_type shape

```
if rdata['selector'] == 0:  
    certdata = cert.as_der()  
elif rdata['selector'] == 1:  
    certdata = cert.get_pubkey().as_der()  
else:  
    raise ValueError('Unknown selector')  
  
if rdata['matching_type'] == 1:  
    certdata = hashlib.sha256(certdata).digest()  
elif rdata['matching_type'] == 2:  
    certdata = hashlib.sha512(certdata).digest()  
else:  
    raise ValueError('Unknown matching type')
```

- And see if certdata matches TLSA certificate association data
- With usages PKIX-TA (0) and PKIX-EE (1)
we need to PKIX validate too (i.e. `connection.verify_ok()`)

```
if str(certdata) == str(rdata['certificate_association_data'])\
and (rdata['certificate_usage'] > 1 or connection.verify_ok()):
    TLSA_matched = True
    print('DANE validated successfully')
    break # from "for tlsa in tlssas:" (first one wins!)

if not TLSA_matched:
    print('DANE validation failed')
```

Hands
on



DANE authenticated
TLS connect (*python*)

- Our DANE example in action:

```
willem@bonobo:~/jcsa17$ ./dane-connect.py www.afnic.fr
Connecting to 2001:67c:2218:30::24
DANE validated successfully
Connecting to 192.134.5.24
DANE validated successfully
```

```
willem@bonobo:~/jcsa17$ ./dane-connect.py www.sidn.nl
Connecting to 2001:7b8:606:294::3
DANE validated successfully
Connecting to 212.114.98.233
DANE validated successfully
```

```
willem@bonobo:~/jcsa17$ ./dane-connect.py www.nominet.uk
Connecting to 2400:cb00:2048:1::6814:e3d
No TLSAS. Regular PKIX validation succeeded
Connecting to 104.20.14.61
No TLSAS. Regular PKIX validation succeeded
```

Hands on C API – How to do a simple query

- Use API and non-API functions
- Do a synchronous request (one at a time).
- Extract data from the response dict
- Do asynchronously requests
- Use an event loop libraries
- Use extension dictionaries
- <https://getdnsapi.net/blog/simple-lookup/>
- <https://getdnsapi.net/static/getdns-jcsa17-examples-0.1.0.tar.gz>
- <https://github.com/getdnsapi/getdns-jcsa17>

Hands on



C API – How to do a simple query **setup context**

```
#include <getdns/getdns_extra.h>
#include <stdio.h>
```

```
int main()
{
    getdns_return_t r;
    getdns_context_t *ctxt = NULL;

    if ((r = getdns_context_create(&ctxt, 1)))
        fprintf( stderr, "Could not create context: %s\n"
                 , getdns_get_errorstr_by_id(r));

    if (ctxt)
        getdns_context_destroy(ctxt);

    return r ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

- GETDNS_RETURN_SUCCESS == 0
- `getdns_get_errorstr_by_id()` is a non-API function
- `getdns_context_create()` does not touch `ctxt` on failure

Hands
on



C API – How to do
a simple query
do query

```
#include <getdns/getdns_extra.h>
#include <stdio.h>
```

```
int main()
{
    getdns_return_t r;
    getdns_context *ctxt = NULL;
    getdns_dict *resp = NULL;

    if ((r = getdns_context_create(&ctxt, 1)))
        fprintf(stderr, "Could not create context: %s\n"
                , getdns_get_errorstr_by_id(r));

    else if ((r = getdns_address_sync(ctxt, "getdnsapi.net.", NULL, &resp)))
        fprintf(stderr, "Unable to do an address lookup: %s\n"
                , getdns_get_errorstr_by_id(r));

    if (resp)
        getdns_dict_destroy(resp);
    if (ctxt)
        getdns_context_destroy(ctxt);

    return r ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Hands on



C API – How to do a simple query do query

```
#include <getdns/getdns_extra.h>
#include <stdio.h>
```

```
int main()
{
    getdns_context_t ctxt;
    getdns_address_t resp;
    if ((r = getdns_context_create(&ctxt)) != GETDNS_EOK) {
        if (r == GETDNS_EADDRINUSE)
            getdns_dict_destroy(resp);
        else
            ; /* error handling */
        getdns_context_destroy(ctxt);
    } else
        ; /* error handling */

    if (r == GETDNS_EOK)
        if ((r = getdns_address_sync(ctxt, ..., &resp)) != GETDNS_EOK)
            getdns_dict_destroy(resp);

    return r ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Hands
on



C API – How to do a simple query do query

```
#include <getdns/getdns_extra.h>
#include <stdio.h>

int main()
{
    getdns_context_t ctxt;
    getdns_address_t resp;
    getdns_error_t r;

    if ((r = getdns_context_create(&ctxt, 1)))
        goto escape;

    if ((r = getdns_address_sync(ctxt, ..., &resp)))
        goto escape_destroy_context;

    if ((r = getdns_something(...)))
        goto escape_destroy_resp;

    escape_destroy_resp:
        getdns_dict_destroy(resp);

    escape_destroy_context:
        getdns_context_destroy(ctxt);

    if (r)
        escape:
            return r ? EXIT_FAILURE : EXIT_SUCCESS;

    return 0;
}
```

Hands on



C API – How to do a simple query do query

```
#include <getdns/getdns_extra.h>
#include <stdio.h>
```

```
int main()
{
    // Dashed green line starts here
    if (!r = getdns_context_create(&ctxt, 1))
        && !(r = getdns_address_sync(ctxt, ..., &resp))
        && !(r = getdns_something(...))
        && !(r = getdns_something_else(...))) {

        /* The happy path */
    } else
        fprintf( stderr, "Something went wrong somewhere: %s\n"
                , getdns_get_errorstr_by_id(r));

    if (resp) getdns_dict_destroy(resp);
    if (ctxt) getdns_dict_destroy(ctxt);

    return r ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Hands
on



C API – How to do a simple query get data

```
getdns_bindata *address;
char address_str[1024];

if ((r = getdns_context_create(&ctxt, 1)))
    fprintf(stderr, "Could not create context: %s\n"
            , getdns_get_errorstr_by_id(r));

else if ((r = getdns_address_sync(ctxt, "getdnsapi.net.", NULL, &resp)))
    fprintf(stderr, "Unable to do an address lookup: %s\n"
            , getdns_get_errorstr_by_id(r));

else if ((r = getdns_dict_get_bindata( resp,
                                       "/just_address_answers/0/address_data", &address)))
    fprintf(stderr, "Unable to get an address from the response: %s\n"
            , getdns_get_errorstr_by_id(r));

else if (address->size != 4 && address->size != 16)
    fprintf(stderr, "Unable to determine type of this address\n");

else if (! inet_ntop( address->size == 4 ? AF_INET : AF_INET6
                     , address->data, address_str, sizeof(address_str)))
    fprintf(stderr, "Could not convert address to string\n");
else
    printf("An address of getdnsapi.net is: %s\n", address_str);
```

Hands
on



C API – How to do a simple query get data

```
getdns_bindata *address;  
char address_str[1024];
```

```
if ((r = getdns_context_create(&ctxt, 1))  
    &amp; (r = getdns_address_sync(ctxt, "getdnsapi.net.", NULL, &resp)))  
else if ((r = getdns_get_errorstr_by_id(r));
```

- JSON-pointer introduced into the API by us
- *getdns* returns "network" format IPv4 and IPv6 addresses
- No data is converted, response dicts brings you to the spot

```
else if ((r = getdns_dict_get_bindata( resp,  
    "/just_address_answers/0/address_data", &address)))  
    fprintf(stderr, "Unable to get an address from the response: %s\n"  
        , getdns_get_errorstr_by_id(r));  
  
else if (address->size != 4 && address->size != 16)  
    fprintf(stderr, "Unable to determine type of this address\n");  
  
else if (! inet_ntop( address->size == 4 ? AF_INET : AF_INET6  
                    , address->data, address_str, sizeof(address_str)))  
    fprintf(stderr, "Could not convert address to string\n");  
else  
    printf("An address of getdnsapi.net is: %s\n", address_str);
```

Hands on



C API – How to do a simple query **asynchronous**

```
#include <getdns/getdns_extra.h>
#include <stdio.h>

void callback(getdns_context *ctxt, getdns_callback_type_t cb_type,
    getdns_dict *resp, void *userarg, getdns_transaction_t trans_id) {}

int main()
{
    getdns_return_t r;
    getdns_context *ctxt = NULL;

    if ((r = getdns_context_create(&ctxt, 1)))
        fprintf(stderr, "Could not create context: %s\n"
            , getdns_get_errorstr_by_id(r));

    else if ((r = getdns_address(ctxt, "getdnsapi.net.", 0, 0, 0, callback)))
        fprintf(stderr, "Unable to schedule an address lookup: %s\n"
            , getdns_get_errorstr_by_id(r));
    else
        getdns_context_run(ctxt);

    if (ctxt)
        getdns_context_destroy(ctxt);
    return r ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Hands on



C API – How to do a simple query **asynchronous**

```
#include <getdns/getdns_extra.h>
#include <stdio.h>
```

```
void callback(getdns_context *ctxt, getdns_callback_type_t cb_type,
    getdns_dict *resp, void *userarg, getdns_transaction_t trans_id) {}
```

```
int main()
```

```
{
```

- A request is *scheduled*
- A callback function is *registered*
- `getdns_context_run()` is **not** an API function

```
    if ((r = getdns_context_create(&ctxt, 0)) != GETDNS_EOK)
        fprintf(stderr, "Failed to create context: %s\n"
                , getdns_get_errorstr_by_id(r));

```

```
    else if ((r = getdns_address(ctxt, "getdnsapi.net.", 0, 0, 0, callback)))
        fprintf(stderr, "Unable to schedule an address lookup: %s\n"
                , getdns_get_errorstr_by_id(r));
    else
        getdns_context_run(ctxt);

    if (ctxt)
        getdns_context_destroy(ctxt);
    return r ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Hands on



C API – How to do a simple query **async libuv**

```
int main() {
    getdns_return_t r;
    getdns_context *ctxt = NULL;
    uv_loop_t loop;

    if (uv_loop_init(&loop)) {
        fprintf(stderr, "Could not initialize event loop\n");
        return EXIT_FAILURE;
    }
    else if ((r = getdns_context_create(&ctxt, 1)))
        fprintf(stderr, "Could not create context: %s\n"
                , getdns_get_errorstr_by_id(r));

    else if ((r = getdns_extension_set_libuv_loop(ctxt, &loop)))
        fprintf(stderr, "Unable to set the event loop: %s\n"
                , getdns_get_errorstr_by_id(r));

    else if ((r = getdns_address(ctxt, "getdnsapi.net.", 0, 0, 0, callback)))
        fprintf(stderr, "Unable to schedule an address lookup: %s\n"
                , getdns_get_errorstr_by_id(r));
    else
        uv_run(&loop, UV_RUN_DEFAULT);

    if (ctxt) getdns_context_destroy(ctxt);
    return r ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

```
#include <uv.h>
int main() {
    uv_loop_t loop;
    if (uv_loop_init(&loop)) {
        fprintf(stderr, "Could not initialize event loop\n");
        return EXIT_FAILURE;
    }
    else if ((r = getdns_context_create(&ctxt, 1)))
        fprintf(stderr, "Could not create context: %s\n"
                , getdns_get_errorstr_by_id(r));
    else if ((r = getdns_extension_set_libuv_loop(ctxt, &loop)))
        fprintf(stderr, "Unable to set the event loop: %s\n"
                , getdns_get_errorstr_by_id(r));
    else if ((r = getdns_address(ctxt, "getdnsapi.net.", 0, 0, 0, callback)))
        fprintf(stderr, "Unable to schedule an address lookup: %s\n"
                , getdns_get_errorstr_by_id(r));
    else
        uv_run(&loop, UV_RUN_DEFAULT);
    if (ctxt) getdns_context_destroy(ctxt);
    return r ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Hands
on



C API – How to do a simple query get data 2

```
void callback(getdns_context *ctxt, getdns_callback_type_t cb_type,
    getdns_dict *resp, void *userarg, getdns_transaction_t trans_id)
{
    getdns_return_t r;
    getdns_list    *jaa;           /* The just_address_answers list */
    size_t          i;             /* Variable to iterate over the jaa list */
    getdns_dict    *ad;            /* A dictionary containing an address */

    if (cb_type != GETDNS_CALLBACK_COMPLETE)
        fprintf(stderr, "Something went wrong with this query: %s\n"
            , getdns_get_errorstr_by_id(cb_type));

    else if ((r = getdns_dict_get_list(resp, "just_address_answers", &jaa)))
        fprintf(stderr, "No addresses in the response dict: %s\n"
            , getdns_get_errorstr_by_id(r));

    else for (i = 0; !getdns_list_get_dict(jaa, i, &ad); i++) {

        getdns_bindata *address;
        char           address_str[1024];

        if ((r = getdns_dict_get_bindata(ad, "address_data", &address)))
            fprintf(stderr, "Could not get address_data: %s\n"
                , getdns_get_errorstr_by_id(r));
    }
}
```

Hands
on



C API – How to do a simple query get data 2

```
void callback(getdns_context *ctxt, getdns_callback_type_t cb_type,
    getdns_dict *resp, void *userarg, getdns_transaction_t tr?) {
    getdns_return_t r;
    getdns_list *jaa; /* The just_address_answers list */
    size_t i; /* Variable to iterate over the jaa list */
    getdns_dict *ad; /* A dictionary containing an address */

    if (cb_type != GETDNS_CALLBACK_COMPLETE)
        fprintf(stderr, "Something went wrong with this query: %s\n"
            , getdns_get_errorstr_by_id(cb_type));

    else if ((r = getdns_dict_get_list(resp, "just_address_answers", &jaa)))
        fprintf(stderr, "No addresses in the response dict: %s\n"
            , getdns_get_errorstr_by_id(r));

    else for (i = 0; !getdns_list_get_dict(jaa, i, &ad); i++) {
        getdns_bindata *address;
        char address_str[1024];

        if ((r = getdns_dict_get_bindata(ad, "address_data", &address)))
            fprintf(stderr, "Could not get address_data: %s\n"
                , getdns_get_errorstr_by_id(r));
    }
}
```

works with
all constants

stop for
when != 0

Hands on C API – How to do a simple query get data 2

```
size_t i; /* Variable to iterate over the jaa list */
size_t cb_type; /* Type of callback for awaiting */

if (cb_type != BACK_COMPLETE)
    fprintf(stderr, "Something went wrong with this query: %s\n"
            , getdns_get_errorstr_by_id(cb_type));

else if ((r = getdns_dict_get_list(resp, "just_address_answers", &jaa)))
    fprintf(stderr, "No addresses in the response dict: %s\n"
            , getdns_get_errorstr_by_id(r));

else for (i = 0; !getdns_list_get_dict(jaa, i, &ad); i++) {

    getdns_bindata *address;
    char address_str[1024];

    if ((r = getdns_dict_get_bindata(ad, "address_data", &address)))
        fprintf(stderr, "Could not get address_data: %s\n"
                , getdns_get_errorstr_by_id(r));

    else if (address->size != 4 && address->size != 16)
        fprintf(stderr, "Unable to determine address type\n");

    else if (!inet_ntop( address->size == 4 ? AF_INET : AF_INET6,
                        address->data, address_str, sizeof(address_str)))
        fprintf(stderr, "Could not convert address to string\n");
    else
        printf("An address of getdnsapi.net is: %s\n", address_str);
}

getdns_dict_destroy(resp); /* Safe, because resp is NULL on error */
}
```

Hands on



C API – How to do a simple query multiple queries

```
struct dane_query_st {
    getdns_dict *addrs_response;
    getdns_transaction_t addrs_transaction_id;
    getdns_dict *tlsas_response;
    getdns_transaction_t tlsas_transaction_id;
};

int main()
{
    getdns_return_t r;
    getdns_context *ctxt = NULL;
    uv_loop_t loop;
    getdns_dict *ext;
    struct dane_query_st state = { NULL, 0, NULL, 0 };
    -----
    // -----
    else if ((r = getdns_context_set_resolution_type(
        ctxt, GETDNS_RESOLUTION_STUB)))
        fprintf(stderr, "Could not set stub resolution modus: %s\n"
            , getdns_get_errorstr_by_id(r));

    else if ((r = getdns_address( ctxt, "getdnsapi.net.", NULL
        , &state
        , &state.addrs_transaction_id
        , addresses_callback)))
        fprintf(stderr, "Unable to schedule an address lookup: %s\n"
            , getdns_get_errorstr_by_id(r));
```

void *
userarg

Hands
on



C API – How to do a simple query multiple queries

```
else if (!(ext = getdns_dict_create())) {
    fprintf(stderr, "Could not allocate extensions dict\n");
    r = GETDNS_RETURN_MEMORY_ERROR;
}
else if ((r = getdns_dict_set_int(ext, "dnssec_return_only_secure"
                                  , GETDNS_EXTENSION_TRUE))
         || (r = getdns_dict_set_int(ext, "dnssec_roadblock_avoidance"
                                  , GETDNS_EXTENSION_TRUE)))
    fprintf(stderr, "Could not populate extensions dict: %s\n"
            , getdns_get_errorstr_by_id(r));

else if ((r = getdns_general( ctxt, "_443._tcp.getdnsapi.net."
                            , GETDNS_RRTYPE_TLSA, ext
                            , &state
                            , &state.tlsas_transaction_id
                            , tlsas_callback)))
    fprintf(stderr, "Unable to schedule a TLSA lookup: %s\n"
            , getdns_get_errorstr_by_id(r));
else
    uv_run(&loop, UV_RUN_DEFAULT);
```

void *
userarg

- Create & populate extensions the API way

Hands
on



C API – How to do a simple query multiple queries

```
else if ((r = getdns_str2dict(
    "{ dnssec_return_only_secure : GETDNS_EXTENSION_TRUE "
    ", dnssec_roadblock_avoidance: GETDNS_EXTENSION_TRUE }", &ext)))

    fprintf( stderr, "Could not create/populate extensions dict: %s\n"
        , getdns_get_errorstr_by_id(r));

else if ((r = getdns_general( ctxt, "_443._tcp.getdnsapi.net."
    , GETDNS_RRTYPE_TLSA, ext
    , &state
    , &state.tlsas_transaction_id
    , tlsas_callback)))
    fprintf( stderr, "Unable to schedule a TLSA lookup: %s\n"
        , getdns_get_errorstr_by_id(r));

else
    uv_run(&loop, UV_RUN_DEFAULT);
```

void *
userarg

- Create & populate extensions the unofficial non-API way

Hands
on



C API – How to do a simple query multiple queries

```
void addresses_callback(getdns_context *ctxt, getdns_callback_type_t cb_type,
    getdns_dict *resp, void *userarg, getdns_transaction_t trans_id)
{
    struct dane_query_st *state = (struct dane_query_st *)userarg;

    if (cb_type != GETDNS_CALLBACK_COMPLETE) {
        /* Something went wrong,
         * Cancel the TLSA query if it hasn't finished yet.
         * Then abort the connection.
        */
        if (! state->tlsas_response)
            (void) getdns_cancel_callback(
                ctxt, state->tlsas_transaction_id);

        abort_connection(state);
        return;
    }
    state->addrs_response = resp;
    if (state->tlsas_response)
        setup_connection(state);
    else
        ; /* Wait for TLSA lookup to complete */
}
```

Hands
on



C API – How to do a simple query multiple queries

```
void tlsas_callback(getdns_context *ctxt, getdns_callback_type_t cb_type,
    getdns_dict *resp, void *userarg, getdns_transaction_t trans_id)
{
    struct dane_query_st *state = (struct dane_query_st *)userarg;

    if (cb_type != GETDNS_CALLBACK_COMPLETE) {
        /* Something went wrong,
         * Cancel the TLSA query if it hasn't finished yet.
         * Then abort the connection.
        */
        if (! state->addrs_response)
            (void) getdns_cancel_callback(
                ctxt, state->addrs_transaction_id);

        abort_connection(state);
        return;
    }
    state->tlsas_response = resp;
    if (state->addrs_response)
        setup_connection(state);
    else
        ; /* Wait for address lookup to complete */
}
```

Hands
on



C API – How to do
a simple query
multiple queries

```
void abort_connection(struct dane_query_st *state)
{
    getdns_dict_destroy(state->addrs_response);
    getdns_dict_destroy(state->tlsas_response);
    fprintf(stderr, "DNS failure\n");
}

void setup_connection(struct dane_query_st *state)
{
    uint32_t status;

    if (getdns_dict_get_int(state->tlsas_response, "status", &status)
        || status == GETDNS_RESPSTATUS_ALL_BOGUS_ANSWERS) {

        abort_connection(state);
        return;
    }
    printf("DNS lookups were successful!\n");

    /* Schedule opening the TLS connection to the addresses (if any)
     * and verification with the received TLSAs (if any)
     * i.e. uv_tcp_connect(connect, socket, dest, callback);
     */
}
```