# getdns API

Verisign Labs, NLNet Labs, No Mountain Software, Sinodun

## What is getdns?

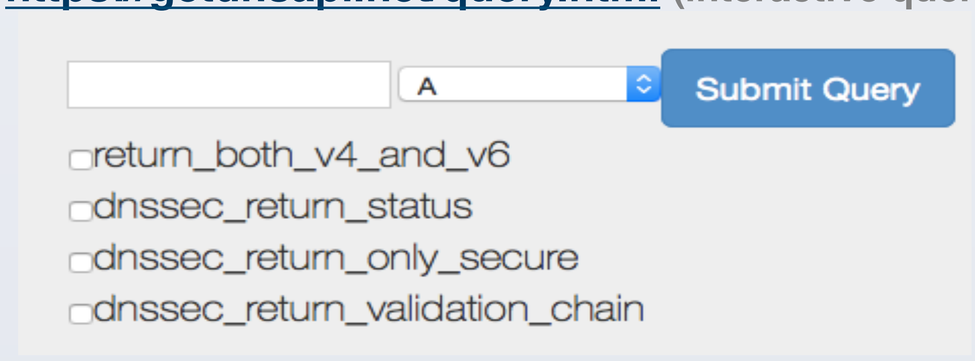
getdns is a modern, application friendly DNS interface that can be used by programmers to more easily query arbitrary data in the DNS. Features include:

- Asynchronous & synchronous operation
- Validating recursive & stub resolver modes
- DNSSEC and DANE support
- Full DNSSEC validation chain acquisition
- Advanced TCP option setting: persistent connections, pipelining & out-of-order processing, Fast Open
- Support for DNS over TLS (work In progress)

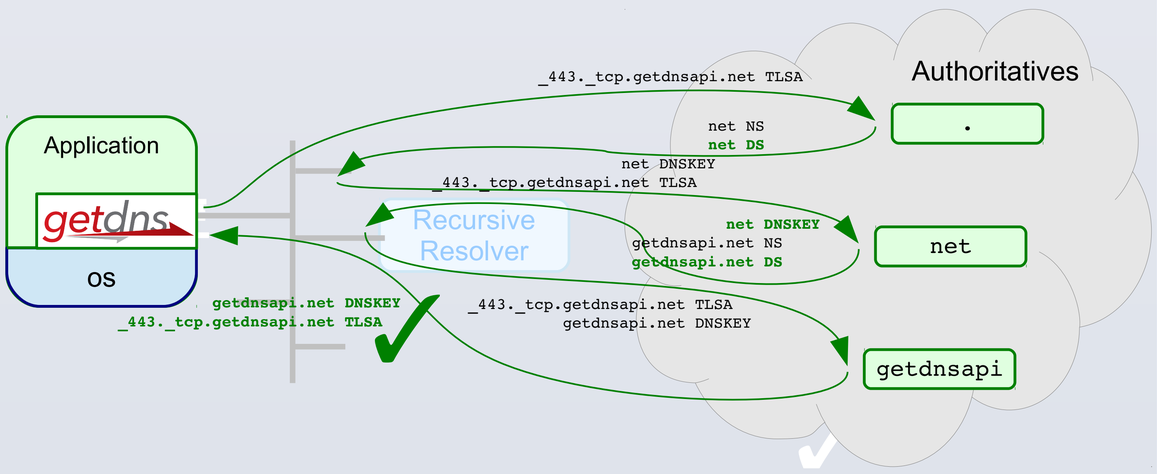
## Our getdns implementation

C Library with bindings in Python, Node.js (and others in the pipeline: Java, PHP, Ruby, etc.):

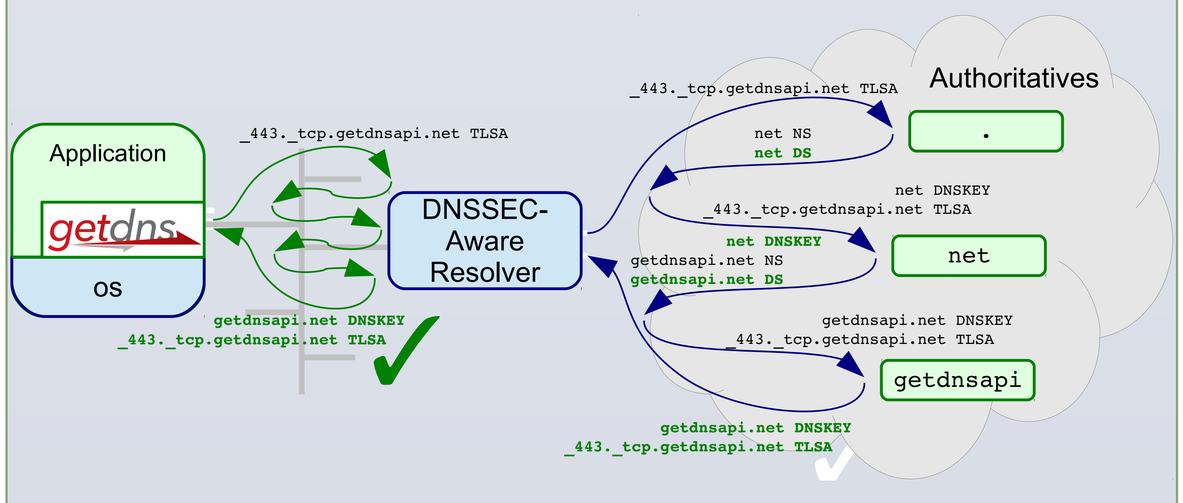
- **https://getdnsapi.net/ (main website)**
- **https://getdnsapi.net/spec.html (specification)**
- **https://getdnsapi.net/query.html (interactive query)**




## Full Recursive Resolver mode




## Validating Stub Resolver mode




## Node.js: efficiently query and setup DANE

```
/* Demonstration of leveraging getdns asynchronous capabilities
 * to setup a DANE connection efficiently.
 *
 * Example output:
 *
 * looking up addresses for getdnsapi.net
 * looking up TLSA for getdnsapi.net for TCP port 443
 * lookups dispatched...
 * lookup of addresses for getdnsapi.net finished
 * setting up tls connection with 185.49.141.37
 * tls setup dispatched
 * setting up tls finished
 * lookup of TLSAs for getdnsapi.net finished
 * Comparing TLSAs with certificate...
 * exit
 */
var tls = require('tls');
var getdns = require('getdns');

function verify_tlsa(conn, err, res)
{
        if (err) {
                console.log( err );
                return
        }
        if (res) {
                console.log('lookup of TLSAs for ' + conn.name + ' finished');
                conn.tlsa_rrs =  res.replies_tree[0].answer
        }
        if (conn.socket && conn.tlsa_rrs && !conn.tlsa_verified) {
                var cert = conn.socket.getPeerCertificate();
                console.log( 'Comparing TLSAs with certificate...' );
                for (var tlsa_rr in conn.tlsa_rrs)
                        // Try to match with cert...
                        ;
        }
}

function setup_tls(conn, err, res)
{
        console.log('lookup of addresses for ' + conn.name + ' finished');
        if (res) {
                console.log( 'setting up tls connection with '
                           + res.just_address_answers[0]);
                conn.socket = tls.connect( 443
                                         , { host: res.just_address_answers[0]
                                           , rejectUnauthorized: false
                                           , servername:conn.name
                                           }
                                         , function() {
                                             console.log( 'setting up tls '
                                                        + 'finished');
                                             verify_tlsa(conn, null, null)
                                           }
                                         );
                console.log('tls setup dispatched')
        } else if (err)
                console.log(err)
}

ctx = getdns.createContext();

var conn = { name         : 'getdnsapi.net'
           , socket       : null
           , tlsa_rrs     : null
           , tlsa_verified: false
           };

console.log( 'looking up addresses for ' + conn.name );
ctx.address( 'getdnsapi.net'
           , function(err, res) { setup_tls(conn, err, res) });

console.log( 'looking up TLSA for ' + conn.name + ' for TCP port 443');
ctx.general( '_443._tcp.getdnsapi.net', getdns.RRTYPE_TLSA
           , { dnssec_return_only_secure: true }
           , function(err, res) { verify_tlsa(conn, err, res) });

console.log( 'lookups dispatched...' );

process.on('exit', function() { console.log('exit'); ctx.destroy() })
```

## Python: implement DNS cookies

```
#!/usr/bin/env python

# Example implementation of the Simple DNS Cookie Option described in
# sections 6 and 7 of draft-ietf-dnsop-cookies-01.txt
#
# For the demo we run a open resolver on demo-ns.getdnsapi.net that only
# answers when provided a valid cookie, or when the request came over TCP,
# and returns REFUSED with the TC bit set otherwise.
#
import getdns, time, random, hashlib

COOKIE_OPCODE = 65001

class CookieStub:
        secret = ''
        update_secret = 0
        secret_lifetime = 5 * 60 * 60 # Refresh secret every 5 hours

        def __init__(self, upstream, secret_lifetime = CookieStub.secret_lifetime ):
                self.upstream = upstream
                self.ctx = getdns.Context()
                self.ctx.resolution_type = getdns.RESOLUTION_STUB
                self.ctx.upstream_recursive_servers = [
                    { 'address_data': upstream
                    , 'address_type': 'IPv6' if ':' in upstream else 'IPv4' }]
                self.server_cookie = ''

        def address(self, hostname):
                # Refresh secret if needed
                #
                now = time.time()
                if now > CookieStub.update_secret:
                        # Create new secret
                        #
                        CookieStub.secret= ''.join(
                            [chr(random.randint(0, 255)) for i in range(8)])

                        # Next update over <life_time> with 30% jitter
                        #
                        CookieStub.update_secret = now \
                            + 0.85 * self.secret_lifetime \
                            + random.randint(0, 0.3 * self.secret_lifetime)

                # Calculate client cookie for this upstream and append
                # the last seen server cookie (if any)
                #
                cookie = hashlib.sha256(CookieStub.secret + self.upstream
                                        ).digest()[:8] + self.server_cookie

                # Do the query with the cookie
                #
                extensions = {'add_opt_parameters': {'options':
                    [{ 'option_code': COOKIE_OPCODE, 'option_data': cookie }]
                }}
                r = self.ctx.address(hostname, extensions)

                try:
                        # Get options from the OPT RR in the additional section
                        #
                        opts = [rr for rr in r['replies_tree'][0]['additional']
                                if rr['type'] == getdns.RRTYPE_OPT
                               ][0]['rdata']['options']

                        # Get the received cookie from those options
                        #
                        new_cookie = str([o['option_data'] for o in opts
                                          if o['option_code'] == COOKIE_OPCODE][0])

                        # Store the new server cookie (if our cookie matched)
                        #
                        if cookie[:8] == new_cookie[:8]:
                                self.server_cookie = new_cookie[8:]

                except IndexError:
                        pass
                except KeyError:
                        pass

                return r


# CookieStub with demo-ns.getdnsapi.net as upstream
stub = CookieStub('185.49.141.38')

# The first query stores the cookie (answer over TCP)
stub.address('getdnsapi.net')

# Subsequent queries use a correctserver cookie and stay on UDP
response_dict = stub.address('verisignlabs.com')

print(response_dict['just_address_answers'])
```

March 2015