getdns API is:

- A DNS API specification by and for application developers (for resolving)
- First implementation by VERISIGN LABS and NLnet Labs (for applications)

From Verisign:
- Allison Mankin, Glen Wiley,
- Neel Goyal, Angelique Finan,
- Craig Despeaux, Shumon Huque, Duane Wessels, Gowri Visweswaran

From NLnet Labs:
- Willem Toorop, Wouter Wijngaards, Olaf Kolkman

From No Mountain Software:
- Melinda Shore

From Sinodun:
- John & Sara Dickinson
A DNSSEC enabled resolver protects against cache poisoning

Application does not know an answer is secure (AD bit not given with getaddrinfo())
A DNSSEC enabled resolver protects against cache poisoning
Application does not know an answer is secure
Is the local network resolver trustworthy?
Motivation - DNSSEC - The Last Mile

- A DNSSEC enabled resolver protects against cache poisoning
- Application does not know an answer is secure
- Is the local network resolver trustworthy?
A DNSSEC enabled resolver protects against cache poisoning

Application does not know an answer is secure

Is the local network resolver trustworthy?

Is the local network resolver validating?

(90% of RIPE ATLAS probes have a DNSSEC-aware resolver
Presentation next Wednesday at the DNS-WG of RIPE68)
Motivation - DNSSEC - DANE

- A DNSSEC enabled resolver protects against cache poisoning
- By delivering origin authentication
- Enabling DNS-based Authentication of Named Entities
Motivation - DNSSEC - DANE

- A DNSSEC enabled resolver protects against cache poisoning
- By delivering origin authentication
- Enabling **DNS-based Authentication of Named Entities**

Entities to be used in applications ... a perfect match ...
Motivation - Other DNS data

not accessible with `getaddrinfo()` & `getaddrinfo()`

- TXT lookups for e-mail
  
  *Sender Policy Framework (SPF), Domain Keys Identified Mail (DKIM), Domain-based Message Authentication, Reporting and Conformance (DMARC)*

- SRV lookups for applications using services (jabber, sip etc.)

- DANE lookups
  
  - TLSA for setting up TLS connections
  - SMIMECERT and OPENPGPKEY for validating signed e-mail

  *and to lookup keys for encrypting messages*

- more will follow...
Motivation - for a new DNS API

From API Design considerations:

... 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 ...
Motivation - for a new DNS API

From API Design considerations:

... 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 ...

Goal

... API design from talking to application developers ...

... create a natural follow-on to gettaddrinfo() ...
Motivation - for a new DNS API

Goal

... API design from talking to application developers ... 

... create a natural follow-on to gettaddrinfo() ...

▶ http://www.vpnc.org/getdns-api/
▶ Edited by Paul Hoffman
▶ First publication April 2013
▶ Updated in February 2014
  (after extensive discussion during implementation)
▶ Creative Commons Attribution 3.0 Unported License
Motivation - for our implementation

From the README:

... 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 ...

▶ http://getdnsapi.net/
▶ Collaborative effort of Verisign Labs & NLnet Labs
▶ 0.1.0 release in February 2014, 0.1.1 in March
▶ nodejs and python bindings
▶ Hack battle at The Next Web in Amsterdam in April 2014
▶ BSD 3-Clause License
Implementation - Features

- Resolves names and gives fine-grained access to the response
- Both stub and full recursive modes (recursive by default)
- Asynchronous modus operandi is the default
- Response dict type
  - Easy to inspect: getdns_pretty_print_dict()
  - Maps well to popular modern scripting languages
- Delivers validated DNSSEC even in stub mode (off by default :(
  - Given that the recursive resolver is DNSSEC-aware
  - 90% of RIPE ATLAS probes have a DNSSEC-aware resolver
    Presentation next Wednesday at the DNS-WG of RIPE68
- Modular event base: libevent, libev, libuv
  ... or just use file descriptor
We try to minimize dependencies

**libunbound** For resolving
   (Currently both recursive and stub)

**libldns** For parsing and constructing wire-format DNS data
   (Will do the stub resolving in future releases)

**libidn1** Only `getdns_convert_ulabel_to_alabel()` and `getdns_convert_alabel_to_ulabel()`

Pluggable event library extensions
One or more of: libevent 1, libevent 2, libuv, libev

- Build dependency: doxygen
- Install dependency: unbound-anchor
Implementation - Supported platforms

We support

- Debian 7.0, 7.3
- FreeBSD 8.4, 9.2, 10.0
- RHEL/CentOS 6.4, 6.5
- OSX 10.8, 10.9
- Ubuntu 12.04, 13.10

We provide binary packages for

- CentOS/RHEL 6.5
- MacOS X

Packages are available for

- FreeBSD Via ports
- MacOS X Via homebrew

Packages in the make

- Debian Ondřej Surý
- Fedora Paul Wouters

MS-Windows and Android in the future
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 (+ how to evaluate (not implemented yet)) etc.)
- context contains the resolver cache
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 and resolver cache
- **name** and **request_type** the name and type to lookup
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 and resolver cache
- **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 (not implemented yet)
  - add_opt_parameter (not implemented yet)
Hands on getdns - Async DNS lookups

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 and resolver cache
- 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_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 */
Hands on \texttt{getdns} - Synchronous lookups

\begin{verbatim}
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
);
\end{verbatim}
Hands on getdns - Address lookups

```c
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
);
```

- Also lookups in other name systems (local files, WINS, mDNS, NIS) (not implemented yet)
- With the Domain Name Space returns both IPv4 and IPv6 (i.e. the `return_both_v4_and_v6` extension is set)
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
getdns_return_t getdns_service(
    getdns_context *context,
    const char *name,
    getdns_dict *extensions,
    void *userarg,
    getdns_transaction_t *transaction_id,
    getdns_callback_t callbackfn
);

- Looks up SRV RRs
Hands on getdns - The response object

```json
{
    "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">
        },
        {
            "address_data": <bindata for 2a04:b900:0:100::37>,
            "address_type": <bindata of "IPv6">
        }
    ],
    "replies_full": [
        <bindata of 0x00008180000100020004000103777777...>,
        <bindata of 0x00008180000100020004000903777777...>
    ],
    "replies_tree": [
        { ... first reply ... },
        { ... second reply ... }
    ]
}
```
Hands on getdns - The response object

"replies_tree": [
    { "header": { "qdcount": 1, "ancount": 2, "rd": 1, "ra": 1,
        "opcode": GETDNS_OPCODE_QUERY,
        "rcode": GETDNS_RCODE_NOERROR, ... },

    "question": { "qname": <bindata for www.getdnsapi.net.>,
        "qtype": GETDNS_RRTYPE_A,
        "qclass": GETDNS_RRCLASS_IN, },

    "answer": [ { "name": <bindata for www.getdnsapi.net.>,
        "type": GETDNS_RRTYPE_A,
        "class": GETDNS_RRCLASS_IN,
        "rdata": { "ipv4_address": <bindata for 185.49.141.37>,
            "rdata_raw": <bindata of 0xb9318d25> },
    }, ...,

    "authority": [ ... ],
    "additional": [],
    "canonical_name": <bindata of "www.getdnsapi.net.">,
    "answer_type": GETDNS_NAMETYPE_DNS
    },
    { "header": { ... 

Hands on *getdns* - Data structures

- Data structure types to represent the response object

```c
typedef struct getdns_dict getdns_dict;
typedef struct getdns_list getdns_list;
typedef struct getdns_bindata { size_t size;
    uint8_t *data; } getdns_bindata;
```
Hands on getdns - Data structures

Data structure types to represent the response object

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

And access, create and modify functions for them

getdns_list_get_length(const getdns_list *this_list, size_t *answer);
getdns_list_get_data_type(const getdns_list *this_list, size_t index, getdns_data_type *answer);
getdns_list_get_dict(const getdns_list *this_list, size_t index, getdns_dict **answer);
getdns_list_get_list(const getdns_list *this_list, size_t index, getdns_list **answer);
getdns_list_get_bindata(const getdns_list *this_list, size_t index, getdns_bindata **answer);
getdns_list_get_int(const getdns_list *this_list, size_t index, uint32_t *answer);
getdns_dict_get_names(const getdns_dict *this_dict, getdns_list **answer);
getdns_dict_get_data_type(const getdns_dict *this_dict, const char *name, getdns_data_type *answer);
getdns_dict_get_dict(const getdns_dict *this_dict, const char *name, getdns_dict **answer);
getdns_dict_get_list(const getdns_dict *this_dict, const char *name, getdns_list **answer);
getdns_dict_get_bindata(const getdns_dict *this_dict, const char *name, getdns_bindata **answer);
getdns_dict_get_int(const getdns_dict *this_dict, const char *name, uint32_t *answer);
void getdns_list_create(getdns_list *this_list);
void getdns_list_destroy(getdns_list *this_list);
getdns_list_set_dict(getdns_list *this_list, size_t index, const getdns_dict *child_dict);
getdns_list_set_list(getdns_list *this_list, size_t index, const getdns_list *child_list);
getdns_list_set_bindata(getdns_list *this_list, size_t index, const getdns_bindata *child_bindata);
getdns_list_set_int(getdns_list *this_list, size_t index, uint32_t child_uint32);
getdns_dict * getdns_dict_create();
void getdns_dict_destroy(getdns_dict *this_dict);
getdns_dict_set_dict(getdns_dict *this_dict, const char *name, const getdns_dict *child_dict);
getdns_dict_set_list(getdns_dict *this_dict, const char *name, const getdns_list *child_list);
getdns_dict_set_bindata(getdns_dict *this_dict, ...
Hands on getdns - Data structures

A bit involved and counter intuitive to use in C

```c
struct getdns_dict *response;
int r = getdns_address_address_sync(context, "www.getdnsapi.net", NULL, &response);
if (r != GETDNS_RETURN_GOOD) goto error;

struct getdns_list *replies_tree;
int r = getdns_dict_get_list(response, "replies_tree", &replies_tree);
if (r != GETDNS_RETURN_GOOD) goto error;

struct getdns_dict *reply;
int r = getdns_list_get_dict(replies_tree, 0, &reply);
if (r != GETDNS_RETURN_GOOD) goto error;

struct getdns_list *answer;
int r = getdns_dict_get_list(reply, "answer", &answer);
if (r != GETDNS_RETURN_GOOD) goto error;
```
Hands on **getdns** - Data structures

- A bit involved and counter intuitive to use in C
- But pythonic

```python
response = getdns.address(context, "www.getdnsapi.net")
answer = response["replies_tree"][0]["answer"]
```

- And javascript works well too...

```javascript
var callback = function(err, response) {
    answer = response.replies_tree[0].answer;
}
context.getAddress("www.getdnsapi.net", callback);
```
Hands on getdns - Data structures

- A bit involved and counter intuitive to use in C
- But pythonic

```python
response = getdns.address(context, "www.getdnsapi.net")
answer = response["replies_tree"][0]["answer"]
```

- And javascript works well too...

```javascript
var callback = function(err, response) {
  answer = response.replies_tree[0].answer;
}
context.getAddress("www.getdnsapi.net", callback);
```

- Python bindings by Melinda Shore
  Get them from https://github.com/getdnsapi/getdns-python-bindings

- Node (javascript) bindings by Neel Goyal
  Get them from https://github.com/getdnsapi/getdns-node

- More bindings will follow ...
Hands on *getdns* - Data structures

- A bit cumbersome and counter intuitive to use in C
- But pythonic
- And javascript works well too...

- Maps well to popular modern scripting languages
- and it provides a uniform grammar
- And the C interface has the virtue of extensibility
- New features don’t need new function prototypes
Hands on getdns - Data structures

http://getdnsapi.net/query.html

```json
{
  "answer_type": GETDNS_NAMETYPE_DNS,
  "canonical_name": "<bindata of "getdnsapi.net.">,
  "just_address_answers": [
    {
      "address_data": "<bindata for 185.49.141.37>,
      "address_type": "<bindata of "IPv4">"
    },
    {
      "address_data": "<bindata for 2a04:b900:0:100::37>,
      "address_type": "<bindata of "IPv6">"
    }
  ]
}
```
Hands on get\texttt{dns} - Getting DNSSEC

\texttt{dnssec\_return\_status}

Returns security assertion. Omits bogus answers

```json
{
  "replies\_tree": [],
  "status": GETDNS\_RESPSTATUS\_NO\_SECURE\_ANSWERS
}
```

"\texttt{dnssec\_status}" can also be GETDNS\_DNSSEC\_SECURE, GETDNS\_DNSSEC\_INSECURE or GETDNS\_DNSSEC\_INDETERMINATE

Thus \textbf{not} GETDNS\_DNSSEC\_BOGUS
Hands on getdns - Getting DNSSEC

dnssec_return_status
Returns security assertion. Omits bogus answers

{ # This is the response object
  "replies_tree": [
    { # This is the first reply
      "dnssec_status": GETDNS_DNSSEC_INSECURE,
    },
    "dnssec_status" can be GETDNS_DNSSEC_SECURE,
    GETDNS_DNSSEC_INSECURE or GETDNS_DNSSEC_INDETERMINATE

dnssec_return_only_secure
The DANE extension
Returns security assertion. Omits bogus and insecure answers

{ # This is the response object
  "replies_tree": [],
  "status" : GETDNS_RESPSTATUS_NO_SECURE_ANSWERS,
Hands on *getdns* - Getting DNSSEC

dnssec_return_status

Returns security assertion. Omits bogus answers

```json
{
  "replies_tree": [
    {
      "dnssec_status": GETDNS_DNSSEC_INSECURE
    }
  ]
}
```

"dnssec_status" can be GETDNS_DNSSEC_SECURE, GETDNS_DNSSEC_INSECURE or GETDNS_DNSSEC_INDETERMINATE

dnssec_return_only_secure

The DANE extension

Returns security assertion. Omits bogus and insecure answers

```json
{
  "replies_tree": [],
  "status": GETDNS_RESPSTATUS_NO_SECURE_ANSWERS,
}
```

dnssec_return_validation_chain

Now "dnssec_status" can also be GETDNS_DNSSEC_BOGUS
Hands on getdns - Getting DNSSEC

dnssec_return_status
Returns security assertion. Omits bogus answers

{ # This is the response object
  "replies_tree": [
    { # This is the first reply
      "dnssec_status": GETDNS_DNSSEC_INSECURE,
    }]
  "dnssec_status" can be GETDNS_DNSSEC_SECURE,
  GETDNS_DNSSEC_INSECURE or GETDNS_DNSSEC_INDETERMINATE

dnssec_return_only_secure
Returns security assertion. Omits bogus and insecure answers

{ # This is the response object
  "replies_tree": [],
  "status": GETDNS_RESPSTATUS_NO_SECURE_ANSWERS,
}

dnssec_return_validation_chain
Now "dnssec_status" can also be GETDNS_DNSSEC_BOGUS

getdns_context_set_return_dnssec_status(context) Just us :(
Hands on **getdns** - Get the validation chain

- **dnssec_return_validation_chain** extension:

```json
{  # 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, ... }, ... },
    ]
}
```

- Feed to `getdns` to validate the chain

```c
getdns_validate_dnssec(getdns_list *to_validate, // The answer
                       getdns_list *support_records, // The "validation_chain"
                       getdns_list *trust_anchors // getdns_root_trust_anchor()
);
```
Hands on getdns - Get the validation chain

- **dnssec_return_validation_chain** extension:

  ```
  { # 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, ... }, ... }],
  }
  ```

- Feed to **getdns_validate_dnssec** to validate the chain

  ```
  getdns_return_t
  getdns_validate_dnssec(
    getdns_list *to_validate,   // The answer
    getdns_list *support_records, // The "validation_chain"
    getdns_list *trust_anchors   // getdns_root_trust_anchor()
  );
  ```

  Look for yourself at [http://getdnsapi.net/query.html](http://getdnsapi.net/query.html)
Hands on \textit{getdns} - Asynchronous

From API 1.8 Event-driven Programs

\textit{Each implementation of the DNS API will specify an extension function that tells the DNS context which event base is being used.}

\begin{verbatim}
libevent

#include <getdns/getdns_ext_libevent.h>

use getdns_extension_set_libevent_base(context, base);

link -lgetdns -lgetdns_ext_event

struct event_base *base = event_base_new();

getdns_extension_set_libevent_base(context, base);

getdns_address(context, "getdnsapi.net", NULL, NULL, NULL, NULL, callback);

event_base_dispatch(base);

event_base_free(base);
\end{verbatim}
Hands on \textit{getdns} - Asynchronous

From API 1.8 Event-driven Programs

*Each implementation of the DNS API will specify an extension function that tells the DNS context which event base is being used.*

\textbf{libevent}

\texttt{include} \#include <getdns/getdns\_ext\_libevent.h>  
\texttt{use} getdns\_extension\_set\_libevent\_base(context, base);  
\texttt{link} -lgetdns -lgetdns\_ext\_event

\textbf{libev}

\texttt{include} \#include <getdns/getdns\_ext\_libev.h>  
\texttt{use} getdns\_extension\_set\_libev\_loop(context, loop);  
\texttt{link} -lgetdns -lgetdns\_ext\_ev

\textbf{libuv}

\texttt{include} \#include <getdns/getdns\_ext\_libuv.h>  
\texttt{use} getdns\_extension\_set\_libuv\_loop(context, loop);  
\texttt{link} -lgetdns -lgetdns\_ext\_uv
Hands on getdns - Asynchronous

From API 1.8 Event-driven Programs

*Each implementation of the DNS API will specify an extension function that tells the DNS context which event base is being used.*

```c
int getdns_context_fd(getdns_context *context)

/* Call the event loop */
while (getdns_context_get_num_pending_requests(context, &tv)) {
    int fd = getdns_context_fd(context);
    fd_set read_fds;
    FD_ZERO(&read_fds);
    FD_SET(fd, &read_fds);
    select(fd + 1, &read_fds, NULL, NULL, &tv);
    if (getdns_context_process_async(context) != GETDNS_RETURN)
        // context destroyed
        break;
}
```
**Road map**

**Missing extensions**

- `specify_class`, `add_warning_for_bad_dns`, `return_call_debugging`, `add_opt_parameter`

To get to hop-to-hop controls (i.e. EDNS0), stub needs to be replaced with ldns resolver.

John & Sara Dickinson on board to help with the effort.

Deadline: 20 July (before IETF90)

**More language bindings, more platforms, more name systems**

- Perl, Ruby
- MS-Windows, Android
- local files, WINS, mDNS, NIS

**More C-like C-interface** (specific for our implementation)

**Optimizations**

- Current data structures are build and need a lot of mallocs
- On top of the data structures of ldns
- Accessor functions on the wire data + Just In Time parsing
Gives applications DNSSEC

website  http://getdnsapi.net
github repo  http://github.nl/getdnsapi/getdns
python repo  http://github.nl/getdnsapi/getdns-python-bindings
node repo  http://github.nl/getdnsapi/getdns-node
mailing-list  http://getdnsapi.net/mailman/listinfo/users
API website  http://www.vpnc.org/getdns-api
API list  http://www.vpnc.org/mailman/listinfo/getdns-api
me  Willem Toorop <willem@nlnetlabs.nl>

Application will bootstrap encrypted channels with DANE

What is your role in this?