DNSSEC local validation: using the getdns library to verify DNS results in an application

by Philip Homburg
<philip@f-src.phicoh.com>

DNSSEC compared to PKI

DNSSEC has hierarchical trust

- One root signing key
- You have to trust your registrar, registry, and the root
- Fully recursive, trust is fully delegated
- Requires interacting with multiple auth. DNS servers to obtain all signatures
- Many TLDs use 1024 RSA keys. Some TLDs keep them for years without rotation. RSA 1024 can be broken with a big enough budget.

PKI

- Many CAs that all need to be trusted
- Independent of DNS, but if some somebody controls your DNS, a letsencrypt cert is easy to obtain
- Different types of certs: DV and EV. Wildcard or not.
- Not recursive
- Client receives complete certificate chain

Local DNS resolution

1. Run a recursive resolver (for example unbound) locally
   - Limited functionality
   - Fragile, what if /etc/resolv.conf is changed?
2. Use a DNSSEC validating library

Getdns

See: `https://getdnsapi.net/`

- Support for event driven programming
- Support for DNSSEC
- Modelled after scripting languages like python or javascript: output in lists and dictionaries
- Extensible
- (I will mix specification and reference implementation)

Getdns API

```
getdns_return_t getdns_context_create( getdns_context **context, int set_from_o

getdns_return_t getdns_extension_set_libevent_base(struct getdns_context *conte

getdns_return_t getdns_general( getdns_context *context, const char *name, uint
    void *userarg, getdns_transaction_t *transaction_id, getdns_callback_t call
getdns_return_t getdns_general_sync( getdns_context *context, const char *name,
    getdns_dict **response);

/* Lists: get the length, get the data_type of the value at a given
   position, and get the data at a given position */
getdns_return_t getdns_list_get_length(const getdns_list *list, size_t *answer)
getdns_return_t getdns_list_get_data_type(const getdns_list *list, size_t index
getdns_return_t getdns_list_get_dict(const getdns_list *list, size_t index, get
getdns_return_t getdns_list_get_list(const getdns_list *list, size_t index, get
getdns_return_t getdns_list_get_bindata(const getdns_list *list, size_t index,
getdns_return_t getdns_list_get_int(const getdns_list *list, size_t index, uint

/* Dicts: get the list of names, get the data_type of the
   value at a given name, and get the data at a given name */
getdns_return_t getdns_dict_get_names(const getdns_dict *dict, getdns_list **an
getdns_return_t getdns_dict_get_data_type(const getdns_dict *dict, const char *
getdns_return_t getdns_dict_get_dict(const getdns_dict *dict, const char *name,
getdns_return_t getdns_dict_get_list(const getdns_dict *dict, const char *name,
getdns_return_t getdns_dict_get_bindata(const getdns_dict *dict, const char *na
```

Getdns output

For an interactive example:

https://getdnsapi.net/query/


```
{
  "answer_type": GETDNS_NAMETYPE_DNS,
  "canonical_name": <bindata for fosdem.org.>,
  "just_address_answers":
  [ { "address_data": <bindata for 31.22.22.135>, "address_type": <bindata of "
  "replies_full": [ <bindata of 0xe7cd818000010001000000106666f73...> ],
  "replies_tree": [ {
      "additional": [ {
          "do": 0, "extended_rcode": 0,
          "rdata": { "rdata_raw": <bindata of 0x> },
          "type": GETDNS_RRTYPE_OPT, "udp_payload_size": 4096,
          "version": 0, "z": 0
        } ],
      "answer": [ {
          "class": GETDNS_RRCLASS_IN, "name": <bindata for fosdem.org.>,
          "rdata": { "ipv4_address": <bindata for 31.22.22.135>, "rdata_raw": <
          "ttl": 600, "type": GETDNS_RRTYPE_A
        } ],
      "answer_type": GETDNS_NAMETYPE_DNS,
      "authority": []
```

Example 1: SSHFP

- Ever ignored the following warning?

```
The authenticity of host 'example.org' can't be e
ECDSA key fingerprint is SHA256:IJf1C5Og2mLCPaT7j
```

- Where would you look for the fingerprint?
- Why not store that information in DNS (with DNSSEC)

```
$ host -t sshfp playout.hq.phicoh.net
playout.hq.phicoh.net has SSHFP record 1 1 48f2b98d78ae
```

Current in OpenSSH, based on ldns

```
/* Initialize resolver from resolv.conf */
domain = ldns_dname_new_frm_str(hostname);
if ((err = ldns_resolver_new_frm_file(&ldns_res, NULL)) != \
    LDNS_STATUS_OK) {
}

ldns_resolver_set_dnssec(ldns_res, true); /* Use DNSSEC */

/* make query */
pkt = ldns_resolver_query(ldns_res, domain, rdtype, rdclass, LDNS_RD);

/* initialize rrset */
rrset = calloc(1, sizeof(struct rrsetinfo));

rrdata = ldns_pkt_rr_list_by_type(pkt, rdtype, LDNS_SECTION_ANSWER);
rrset->rri_nrdatas = ldns_rr_list_rr_count(rrdata);

/* copy name from answer section */
len = ldns_rdf_size(ldns_rr_owner(ldns_rr_list_rr(rrdata, 0)));
if ((rrset->rri_name = malloc(len)) == NULL) {
}
memcpy(rrset->rri_name,
    ldns_rdf_data(ldns_rr_owner(ldns_rr_list_rr(rrdata, 0))), len);

rrset->rri_rdclass = ldns_rr_get_class(ldns_rr_list_rr(rrdata, 0));
```

Example using Getdns

See: https://github.com/phicoh/openssh-getdns/tree/getdns

```
/* Create the DNS context for this call */
this_ret = getdns_context_create(&this_context, 1);
/* Try to enable roadblock avoidance */
this_ret = getdns_str2dict( "{ dnssec_roadblock_avoidance: GETDNS_EXTENSION
getdns_context_config(this_context, config_dict);
this_extensions = getdns_dict_create();
this_ret = getdns_dict_set_int(this_extensions, "dnssec_return_status", GET

/* Set up the getdns_sync_request call */
this_ret = getdns_general_sync(this_context, hostname, rdtype, this_extensi

/* Be sure the search returned something */
this_ret = getdns_dict_get_int(this_response, "status", &this_error);
if (this_error != GETDNS_RESPSTATUS_GOOD)  // If the search didn't return "

this_ret = getdns_dict_get_list(this_response, "replies_tree", &replies_tre
/* Assume one reply */
this_ret = getdns_list_get_dict(replies_tree_list, 0, &reply_dict);
this_ret = getdns_dict_get_int(reply_dict, "dnssec_status", &dnssec_status)
if (this_ret != GETDNS_RETURN_GOOD)
this_ret = getdns_dict_get_list(reply_dict, "answer", &answer_list);
this_ret = getdns_list_get_length(answer_list, &num_answers);
```

Advantages of Getdns

- Ldns requires the application to set the trust anchor, which ssh does not do. So the code doesn't actually do anything.
- Getdns can also fetch the trust anchor from IANA (signed by ICANN)
- 'Roadblock avoidance': try to work around broken DNS resolvers

Example 2: TLSA

```
$ host -t tlsa _25._tcp.open.nlnetlabs.nl.
_25._tcp.open.nlnetlabs.nl is an alias for 3.1.1._dane-gs.nlnetl
3.1.1._dane-gs.nlnetlabs.nl has TLSA record 3 1 1 544f284d66af2d
```

- TLSA is great for email (SMTP):
  - It informs the sender the receving system supports TLS (no downgrade attacks)
  - It can provide the fingerprint of a self-signed certificate (no need to reply on CAs)

Experimenting With Crypto In Separate Processes

- Isolates from different (TLS) libraries

```
$ tls_helper -?
tls_helper: -c|-s -i <in-fd> [-o <out-fd>]
    [-h <hostname>] [-p <priv-key-file>]
    [-P <cert-file>] [-r <result-fd>]
    [-t <timeout>] [-l]
```

- Avoid shared library issues

```
dns_mx_helper
```

Combine All Relevant DNS Information To Deliver Email

```
$ echo '{ "MX": "fosdem.org" }' | ./dns_mx_helper
{
  "MX": [ { "name": "fosdem.org", "dnssec_secure": True,
    "dnssec_insecure": False, "NXDOMAIN": False,
    "rrset": [ { "pref": 10, "exch": "mail.fosdem.org." } ] } ],
  "A": [ { "name": "mail.fosdem.org.", "dnssec_secure": True,
    "dnssec_insecure": False, "NXDOMAIN": False,
    "rrset": [ { "ipv4_addr": "31.22.22.142" } ] } ],
  "AAAA": [ { "name": "mail.fosdem.org.", "dnssec_secure": True,
    "dnssec_insecure": False, "NXDOMAIN": False,
    "rrset": [ { "ipv6_addr": "2001:67c:1808::142" } ] } ],
  "TLSA": [ { "name": "_25._tcp.mail.fosdem.org.",
    "dnssec_secure": True, "dnssec_insecure": False,
    "NXDOMAIN": True } ]
}
```

And Now With TLSA

```
$ echo '{ "MX": "nlnetlabs.nl" }' | ./dns_mx_helper
{
  "MX": [ { "name": "nlnetlabs.nl", "dnssec_secure": True,
    "dnssec_insecure": False, "NXDOMAIN": False, "rrset": [
    { "pref": 50, "exch": "open.nlnetlabs.nl." } ] } ],
  "A": [ { "name": "open.nlnetlabs.nl.", "dnssec_secure": True,
    "dnssec_insecure": False, "NXDOMAIN": False, "rrset": [
    { "ipv4_addr": "185.49.140.10" } ] } ],
  "AAAA": [ { "name": "open.nlnetlabs.nl.", "dnssec_secure": Tru
    "dnssec_insecure": False, "NXDOMAIN": False, "rrset": [
    { "ipv6_addr": "2a04:b900::1:0:0:10" } ] } ],
  "TLSA": [ { "name": "_25._tcp.open.nlnetlabs.nl.",
    "dnssec_secure": True, "dnssec_insecure": False, "NXDOMAIN":
    { "usage": 3, "sel": 1, "type": 1, "data": "544f284d66af2de0
}
```

Example Code

```
static void init_getdns_context(getdns_context **contextp)
{
        getdns_return_t r;
        getdns_dict     *config_dict = NULL;

        /* Create the DNS context for this call */
        if ((r = getdns_context_create(contextp, 1)))
                fatal("Trying to create the context failed");

        /* Try to enable roadblock avoidance */
        if ((r = getdns_str2dict(
                "{ "
                "dnssec_roadblock_avoidance: GETDNS_EXTENSION_TRUE, "
                "dnssec_return_status: GETDNS_EXTENSION_TRUE, "
                "dnssec_return_validation_chain: GETDNS_EXTENSION_TRUE, "
                "return_call_reporting: GETDNS_EXTENSION_TRUE"
                //"call_reporting: GETDNS_EXTENSION_TRUE"
                " }",
                &config_dict)))
                fatal("trying to create roadblock avoidance dict failed");

        if ((r= getdns_context_config(*contextp, config_dict)) != 0)
        {
                warn("getdns_context_config failed");
        }
```

Example Code (cont,)

```
static struct query_status *do_query(getdns_context *context,
        struct request *request, uint16_t type, char *name)
{
        // variable declarations deleted

        query_status= alloc_query_status(request, type, name, &exists);

        if (!(extensions = getdns_dict_create()))
                fatal("Could not create extensions dict.");
        if ((r = getdns_general_sync(context, name, type,
                extensions, &response)))
                fatal("Error scheduling synchronous request");

        sp= &query_status->status;
        set_status(response, sp);

        /* Done if transient error */
        if (!sp->dnssec_secure && !sp->dnssec_insecure) goto done;
        /* Done if nxdomain or rcode not zero */
        if (sp->nxdomain || sp->rcode != 0) goto done;

        /* Get the answers list */
        if ((r= getdns_dict_get_list(response, "/replies_tree/0/answer", &answe
                fatal("unable to get answer list");
```

Example Code (cont,)

```
static void set_status(getdns_dict *dict, struct status *sp)
{
        if ((r = getdns_dict_get_int(dict, "status", &sp->getdns_status)) != GE
        /* We need to know if we got an NXDOMAIN or not. So continue for
         * a while even if the status is GETDNS_RESPSTATUS_NO_NAME.
         */
        if (sp->getdns_status != GETDNS_RESPSTATUS_GOOD &&
                sp->getdns_status != GETDNS_RESPSTATUS_NO_NAME) { return; }
        if ((r = getdns_dict_get_int(dict, "/replies_tree/0/dnssec_status",
                &dnssec_status)) != GETDNS_RETURN_GOOD) { }
        switch(dnssec_status)
        {
        case GETDNS_DNSSEC_SECURE: sp->dnssec_secure= 1; break;
        case GETDNS_DNSSEC_INSECURE: sp->dnssec_insecure= 1; break;
        case GETDNS_DNSSEC_BOGUS: sp->dnssec_bogus= 1; break;
        case GETDNS_DNSSEC_INDETERMINATE: sp->dnssec_bogus= 1; break;
        default: sp->dnssec_bogus= 1; break;
        }
        /* Continue if we have either dnssec_secure or dnssec_insecure */
        if (!sp->dnssec_secure && !sp->dnssec_insecure) return;
        /* Get rcode */
        if ((r = getdns_dict_get_int(dict, "/replies_tree/0/header/rcode",
                &sp->rcode)) != GETDNS_RETURN_GOOD) { }
        sp->nxdomain= (sp->rcode == GETDNS_RCODE_NXDOMAIN);
}
```

Questions?

- It is worth looking into getdns for any (event-based) DNS code
- The world need more DNSSEC
- How do we kill weak keys? Mark them as bogus in validators?
- DNSSEC and getdns provide interesting new options for key distribution
- Questions? Remarks?

Mail: <philip@f-src.phicoh.com>