DANE & Application Uses of DNSSEC

Shumon Huque, Verisign Labs

Internet2 Technology Exchange, Indianapolis, IN

October 29th, 2014
This session will give an overview of application uses of DNSSEC, available software tools, and application programmer interfaces. With the increasing deployment of DNSSEC, new, exciting uses are emerging that leverage the DNS to store and verify cryptographic keying material (like public keys, certificates, fingerprints, etc). The DANE (DNS-based Authentication of Named Entities) protocol and new DNS records like TLSA are among the principal enablers of these uses. This session will give an overview of DANE in the context of DNSSEC, use cases that it enables today and in the future, and available software tools. It will also talk about a new open DNS API called "getdns" that allows application programmers to more easily use DNS and DNSSEC enhanced responses without needing to be deep experts in the DNS protocol. The Research & Education community has long been a pioneer in deploying new technologies, including DNSSEC. Technologies like DANE are the next step in the evolution of DNSSEC, and we expect that DANE and new software APIs will be adopted and experimented with in this community.
DNSSEC at a glance
DNSSEC at a glance

- Original DNS protocol wasn’t built with security in mind
  - No way to verify the authenticity of DNS data other than trusting the connection to the DNS server
- DNSSEC: “DNS Security Extensions”
- A system to verify the authenticity of DNS data
  - Specifications: RFC 4033, 4034, 4035, 5155
- Protects against DNS spoofing & cache poisoning
- Secondary benefits:
  - Ability to store and verify cryptographic keying material in the DNS, which could be used by new & existing application protocols
  - SSHFP, IPSECKEY, CERT, DKIM, etc.
  - DANE family: TLSA, OPENPGPKEY, SMIMEA, etc.
DNSSEC at a glance

• Uses public key cryptography
• Each zone has a public and private key
  • Typically a 2-level hierarchy (KSK and ZSK) is used for each zone
• Zone owner uses private key to sign the zone data, producing digital signatures for each resource record set
• Public key is used by DNS resolvers to validate the signatures -> proof of authenticity
• Public key is published in the zone
• Zone public keys are organized in a chain of trust that follows the DNS delegation hierarchy
• Resolvers authenticate signatures from the root down to the target zone containing the queried name
Recursive Resolver is prepopulated with root DNS server addresses.

Recursive Resolver

1. Request to www.upenn.edu
2. Referral to .edu
3. Referral to edu
4. Referral to upenn.edu
5. Referral to upenn.edu
6. Answer 1.2.3.4

Endstation (stub resolver)

www.upenn.edu
Recursive Resolver is prepopulated with root DNS server addresses and the root’s public key.

Recursive Resolver (has root’s pubkey)

www.upenn.edu
set DO bit

1

Recursive Resolver

1. (root)

2

referral to .edu
+ DS, RRSIG

3

referral to upenn.edu
+ DS, RRSIG

4

www.upenn.edu

5

set AD bit
+ RRSIG

6

answer 1.2.3.4

7

answer
+ AD bit

8

endstation
(stub resolver)

Recursive Resolver is prepopulated with root DNS server addresses and the root’s public key.

Also queries for DNSKEY and DS records are performed as needed.
## DNSSEC: Additional DNS record types

<table>
<thead>
<tr>
<th>Record Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNSKEY</td>
<td>Contains zone’s public key</td>
</tr>
<tr>
<td>RRSIG</td>
<td>Contains digital signature of record set</td>
</tr>
<tr>
<td>NSEC</td>
<td>Points to next name in the zone (for authenticated denial of existence)</td>
</tr>
<tr>
<td>DS</td>
<td>Delegation signer (signs/authenticates key for a child zone)</td>
</tr>
<tr>
<td>NSEC3</td>
<td>Enhanced version of NSEC (provides zone enumeration defense &amp; opt-out)</td>
</tr>
<tr>
<td>NSEC3PARAM</td>
<td>Parameters for NSEC3 (flags, hash, iterations, salt)</td>
</tr>
</tbody>
</table>
Changes in a signed zone

- One or more DNSKEY records at zone apex
- One or more NSEC for each DNS name
- One or more RRSIG (signature) for each RR Set
- One or more DS records (in parent zone) for every secure delegation to a child zone

- Exceptions: non-authoritative data like delegation NS record sets and glue records do not have signatures
$ dig jabber.upenn.edu A

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 337

;; QUESTION SECTION:
;jabber.upenn.edu. IN A

;; ANSWER SECTION:
jabber.upenn.edu. 86400 IN A 128.91.2.172

;; AUTHORITY SECTION:
upenn.edu. 86400 IN NS sns-pb.isc.org.
upenn.edu. 86400 IN NS adns3.upenn.edu.
upenn.edu. 86400 IN NS adns2.upenn.edu.
upenn.edu. 86400 IN NS adns1.upenn.edu.
upenn.edu. 86400 IN NS dns2.udel.edu.
upenn.edu. 86400 IN NS dns1.udel.edu.

;; ADDITIONAL SECTION:
adns1.upenn.edu. 81904 IN A 128.91.3.128
adns1.upenn.edu. 81904 IN AAAA 2607:f470:1001::1:a
adns2.upenn.edu. 81904 IN A 128.91.254.22
adns2.upenn.edu. 81904 IN AAAA 2607:f470:1002::2:3
adns3.upenn.edu. 81904 IN A 128.91.251.33
adns3.upenn.edu. 81904 IN AAAA 2607:f470:1003::3:c
$ dig jabber.upenn.edu A +dnssec

;; HEADER opcodes: QUERY, status: NOERROR, id: 690
;; flags: qr aa rd
;; OPT PSEUDOSECTION: EDNS: version: 0, flags: do udp: 4096
;; QUESTION SECTION:
;jabber.upenn.edu. IN A

;; ANSWER SECTION:
jabber.upenn.edu. 86400 IN A 128.91.2.172
jabber.upenn.edu. 86400 IN RRSIG A 5 3 86400 (20140909070510 20140810061231 50475 upenn.edu.
  jFElfhdkNeWsIwE2cV/n2nt9T2KXKgYYtyemVEf3X414nbhyXGvFdGETmdS9cV13RgjwwUVaY78jaz7cQPX2lc3raYDrLY3irRh1NSbt9v/esF4SI06KwRmhTv3Z2GBVP+CjFkMLJnN1dFBEa2UHzFzIk7cQcdmEdbiDJ3Ag= )

;; AUTHORITY SECTION:
upenn.edu. 86400 IN NS dns1.udel.edu.
upenn.edu. 86400 IN NS noc3.dccs.upenn.edu.
upenn.edu. 86400 IN NS dns2.udel.edu.
upenn.edu. 86400 IN NS noc2.dccs.upenn.edu.
upenn.edu. 86400 IN RRSIG NS 5 2 86400 20140919232217 (20140819223616 50475 upenn.edu.
WWpT4uD9p5zORM+207pRZ46+Q03cHj9tnjxH62Xt9QBRyu9V7+3ih1IMIHDc9kjsddskT8GJ+5hEzykB8fPIjSlibqG6hCnCccGdTsGzmPoGdlz95H7Nf2yfr1GLAcSCix6IEJb8Aj4+OW9Zq1dmeZrnJDXSzm8joQg5+IlkzR4= )

(Activity 5: RSA-SHA1)
DNSSEC Deployment overview

A very brief overview of DNSSEC deployment in the Internet
Brief DNSSEC Deployment status

- DNS Root was signed in July 2010
- TLDs signed [1]: .COM, .NET, .EDU, .ORG, .GOV, etc.:
  - All TLDs: 543 of 726 (74.8%), as of October 2014
  - ccTLDs: 102 of 286 (36%)
  - New gTLDs: all are signed (418 of 418)
- Reverse trees (in-addr.arpa and ip6.arpa) are signed
- Levels beneath TLDs are where more needs to be done
  - US .GOV federal: ~ 82% [3](Oct 2014) – FISMA OMB Mandate
  - Internet2 Higher Ed members [1]: 27 of ~ 266 (10.2%)
  - .NL (Netherlands) has over 1.8 million signed delegations [2]
  - .COM has 384,100 signed delegations (0.33%) [4]

Use of DNSSEC Validation by resolvers

- US Gov FISMA IT security policy DNSSEC validation mandate (Spring 2014)
- Some very large DNS resolver services are doing DNSSEC validation:
  - Google Public DNS (free; very widely used)
  - Comcast DNS \(^1\): ~ 18.1 million subscriber homes
- A number of US R&E institutions and NRENs
- Worldwide there is substantial amount of validation, as measured by APNIC
  - Roughly \(\frac{1}{4}\) of DNS queries by resolvers in the US perform DNSSEC validation

DNSSEC Validation map (from APNIC)

US
Validating: 24.21%

gronggrong.rand.apnic.net/cgi-bin/worldmap
Application Uses of DNSSEC
Application uses of DNSSEC

- One of the more exciting prospects for DNSSEC
- DNSSEC can be employed to store cryptographic keys in the DNS, and ..
- Allow applications to securely obtain (authenticate) those keys and use them in application security protocols
- Some possible applications: SSH, SSL/TLS, HTTPS, S/MIME, PGP, SMTP, DKIM, and many others ..
- Existing records:
  - SSHFP, IPSECKEY, DKIM TXT record, …
  - DANE records: TLSA, OPENPGPLKEY
- Upcoming:
  - SMIMEA, IPSECA, …
SSHFP record

- Secure Shell Host Key Fingerprint (RFC 4255)
- Allows you to validate SSH host keys using DNSSEC

grodd.magpi.net. 86400 IN SSHFP (1 1 F60AE0994C0B02545D444F7996088E9EA7359CBA)

In OpenSSH, you can use the client configuration directive "VerifyHostKeyDNS" to use this. Enabled by default in some newer operating systems like FreeBSD 10.
**IPSECKEY record**

- RFC 4025: method for storing IPsec keying material in DNS

- rdata format: precedence, gateway-type, algorithm, gateway address, public key

- Not much uptake of this record

- Will likely be superseded by newer proposals, like IPSECA

```
38.2.0.192.in-addr.arpa. 7200 IN IPSECKEY ( 10 1 2 192.0.2.38 AQNRU3mG7TVTO2BkR47usntb102uFJtugbo6BSGvgqt4AQ== )
```
TLS and the Internet PKI

- A very large number of security protocols authenticate server names with X.509 certificates
  - TLS, IPsec, HTTPS, SIPS, SMTP, IMAP, XMPP, …
- These certificates are issued and signed by the Internet PKI, composed of a set of globally trusted public Certification Authorities (CAs)
Public CA model issues

- Applications need to trust a large number of global Certification Authorities (CA)
- No namespace constraints! Any CA can issue certificates for any entity on the Internet
- Least common denominator security: our collective security is equal to the weakest one!
- Furthermore, many of them issue subordinate CA certificates to their customers, again with no naming constraints
- Most CAs aren’t capable of issuing certificates with any but the most basic capabilities (e.g. alternate name forms or other extensions)
Public CA model issues

• “Analysis of the HTTPS Certificate Ecosystem”, UMich, October 2013, Internet Measurement Conference
  • Over 1,800 separate CAs are capable of issuing certificates for anyone! (Root CAs and intermediate CAs issued by them)

• “The Shape & Size of Threats: Defining a Networked System’s Attack Surface”
  • Eric Osterweil (Verisign), Danny McPherson (Verisign), Lixia Zhang (UCLA), NPsec 2014 conference
Can DNSSEC help?

- Can we leverage DNSSEC to address these deficiencies?
- DNS has hierarchical, decentralized administration
- Certificates and public keys placed in the DNS can be authenticated with DNSSEC signatures
- Name constraints are inherent
- Deployed infrastructure is becoming real
- Root and many of the TLDs are signed, so most organizations can sign their zones and have an intact secure chain of trust to the root
- Validation is also becoming more prevalent (see prior slides in deployment status)
DANE and the TLSA record

- RFC 6698: The **DNS-based Authentication of Named Entities (DANE)** Protocol for Transport Layer Security

- Defines a new DNS record type “**TLSA**”, that can be used for better & more secure ways to authenticate SSL/TLS certificates
  - By specifying constraints on which CA can vouch for a certificate, or which specific PKIX end-entity certificate is valid
  - By specifying that a service certificate or a CA can be directly authenticated in the DNS itself.
TLSA record example

ポート、トランスポートプロトコルとサーバーのドメイン名

_TLSA.

443._tcp.www.example.com. IN TLSA ( 0 0 1 d2abde240d7cd3ee6b4b28c54df034b9 7983a1d16e8a410e4561cb106618e971 )

使用
セレクタ
一致タイプ
TLSA rrtype
証明書の関連データ
TLSA configuration parameters

**Usage field:**
- 0  PKIX-TA: CA Constraint
- 1  PKIX-EE: Service Certificate Constraint
- 2  DANE-TA: Trust Anchor Assertion
- 3  DANE-EE: Domain Issued Certificate

**Selector field:**
- 0  Match full certificate
- 1  Match only SubjectPublicKeyInfo

**Matching type field:**
- 0  Exact match on selected content
- 1  SHA-256 hash of selected content
- 2  SHA-512 hash of selected content

Certificate Association Data: raw cert data in hex
TLSA configuration parameters

**Usage field:**
- 0  PKIX-TA: CA Constraint
- 1  PKIX-EE: Service Certificate Constraint
- 2  DANE-TA: Trust Anchor Assertion
- 3  DANE-EE: Domain Issued Certificate

**Selector field:**
- 0  Match full certificate
- 1  Match only SubjectPublicKeyInfo

**Matching type field:**
- 0  Exact match on selected content
- 1  SHA-256 hash of selected content
- 2  SHA-512 hash of selected content

Certificate Association Data: raw cert data in hex
Usage types

0 PKIX-TA: CA Constraint
Specify which CA should be trusted to authenticate the certificate for the service. Full PKIX certificate chain validation needs to be performed.

1 PKIX-EE: Service Certificate Constraint
Define which specific service certificate ("EE cert") should be trusted for the service. Full PKIX cert validation needs to be performed.

2 DANE-TA: Trust Anchor Assertion
Specify a domain operated CA which should be trusted independently to vouch for the service certificate.

3 DANE-EE: Domain Issued Certificate
Define a specific service certificate for the service at this domain name.
Example TLSA record (for WWW)

_443._tcp.fedoraproject.org. 263 IN TLSA 0 0 1 (
  19400BE5B7A31FB733917700789D2F0A2471C0C9D506
  C0E504C06C16D7CB17C0 )

_443._tcp.fedoraproject.org. 263 IN RRSIG TLSA 5 4 300 (  
  20141114150617 20141015150617 7725  
  fedoraproject.org.  
  hrk0si7I/BWTz0wEtMcFZNUCj/0o5796k5FVuZx6eXrc
  YOe/ChHA/Shu/WHR3iM1yNGi86+8t4wMq9GA+JZthWZC
  ZmENxf9OTNe/t/LBAc2EDW/fMBJq0JO2b4ZkJHXCEyX0
  CDsIYz8shZ20nPGlrsYqwLdQiCeravWcwcJiPuc= )

Usage 0 (“CA Constraint”) — this record says:
- For service at fedoraproject.org tcp port 443
- only the CA with the specified SHA-256 certificate fingerprint (19400BE5B...) should be trusted
Enter DANE-TLS

- DNS-Enabled Authentication of Named Entities (DANE)
  RFC6698
DANE/TLSA tools and software

- TLSA Record Generation
  - Command line tools: “swede”, “hash-slinger”, “ldns-dane”
  - Web based tool: https://www.huque.com/bin/gen_tlsa

- TLSA validators for web
  - Some 3rd party validator plugins are available (Firefox, Chrome, Opera, Safari):
    - https://www.dnssec-validator.cz/
  - Bloodhound Mozilla fork:
DANE for SMTP

DANE can be used to help secure (1) and (2)
DANE for SMTP

- DANE in conjunction with SMTP over TLS, or SMTP + STARTTLS can be used to more fully secure email delivery
- DANE can authenticate the certificate of the SMTP submission server that the user’s mail client (MUA) communicates with
- DANE can authenticate TLS connections between SMTP servers (“MTA”s or Mail Transfer Agents)
- This second use case is where DANE solves some important problems that are unaddressed today
DANE for SMTP

• Most connections between SMTP servers today use encryption opportunistically (i.e. if both sides support and advertise it, it is used)

• Even when encryption is used, it is vulnerable to attack:
  • Attackers can strip away the TLS capability advertisement and downgrade the connection to not use TLS
  • TLS connections are often unauthenticated (e.g. the use of self signed certificates as well as mismatched certificates is common)

• DANE can address both these vulnerabilities
  • Authenticate the certificate using a DNSSEC signed TLSA record
  • Use the presence of the TLSA record as an indicator that encryption must be performed (prevent downgrade)

Example TLSA record (for SMTP)

```plaintext
_25._tcp.mx1.freebsd.org. 2389 IN TLSA 3 0 1 ( 5EC0508C3F337D18509F41BFF9D8AB07FED588A132FA 12FA1E223BA6B9403ACB )

_25._tcp.mx1.freebsd.org. 2389 IN RRSIG TLSA 8 5 3600 ( 20141023072418 20141009105807 39939 freebsd.org. 116DEQ7oP2lbEcOeJyPk+I8tYiGz4CzuDiqiMbr4Mzp390UWdej3kdAz4t+1BT0dO3/o0nz0pp3HFsdU+gkwT6YHJg4C6mi3STPciCP1tjbFuW/dv41PkuAaN7kjt/qwPrR60kQmyvcuUoYgUDPbNYbJNJXai+mFai5WqLS2MEP15ydUnt8KympjHS5mVLVGXW0e7tLY1afQz1VrIeYsGW8YztMDYUpCXjWiq+YpCFv7rZ7IcejQR6ot1M35CDsfjk68eu0EAjx+HlqaTdGyilcMB+GduFwqkJULDPIgiFu/3xb+srJRzuR89YpHga9OCnz6nXJgQ6cxvSImZWbKuw== )
```

This is a domain-issued certificate (usage 3), which can be authenticated without a trusted CA.
Early large adopters of SMTP + DANE

Quite a few are large email systems in Germany. See a larger list at [https://www.tlsa.info/](https://www.tlsa.info/)

- posteo.de
- mailbox.org
- umbkw.de
- bund.de
- denic.de
- freebsd.org
- unitybox.de
- debian.org, debian.net
- ietf.org
- nlnetlabs.nl
- nic.cz
- nic.ch
- torproject.org
SMTP servers that support DANE

- Postfix MTA (works today, version 2.11 onwards)
- Exim (currently under development)
XMPP servers

- XMPP (Jabber) has seen some uptake of DANE.
- To authenticate the c2s and/or s2s portion of the XMPP protocol
- List of XMPP servers with DANE TLSA records:
  - https://xmpp.net/reports.php#dnssecdane

Example:

_xmpp-server._tcp.mail.de. 3600 IN SRV 10 20 5269 jabber.mail.de.

_5269._tcp.jabber.mail.de. 600 IN TLSA 3 1 1 (A0315F0CF61CAC787140833C2C608550476 246DDA54122D66BB339D5 0FBB10E3)
OpenPGPKEY

- OPENPGPKEY record
- Used to publish an OpenPGP public keys in the DNS
- DNSSEC signature provides authentication
- Spec under development, but RR code already assigned
Example OPENPGPKEY record

For shuque@huque.com

1\text{st} label: sha224 hash of “shuque” = 4f7c2705c0f139ede60573f8537a0790fb64df5d4a819af951d259bc

2\text{nd} label: “_openpgpkey”

Remaining labels: domain name portion of the email addr: Huque.com

Resulting record looks like this:

4f7c2705c0f139ede60573f8537a0790fb64df5d4a819af951d259bc. _openpgpkey.huque.com. IN OPENPGPKEY <base64 encoding of the openpgp key>
SMIMEA

• Using DNSSEC to associate certificates with domain names for S/MIME

• S/MIME is a method of encrypting and signing MIME data used in e-mail messages

• The SMIMEA DNS record proposes to associate S/MIME certificates with DNS domain names

• Verisign DANE/SMIMEA early Mail User Agent Prototype
getdns: a brief introduction

A new application friendly interface to the DNS
Application access to any kind of DNS data

• Today’s commonly used DNS application interfaces, like `getaddrinfo()`, `getnameinfo()` are designed to obtain the most common types of DNS data, e.g. name to IP address mappings, reverse DNS mappings, etc.

• How do applications ask for other types of data, eg. TLSA, SSHFP records, or even SRV records?

• How can we tell if a response was successfully authenticated with DNSSEC?

• Some lower level, harder to use libraries exist (libresolv etc) that can do some of this, but application developers deserve something much better
Securing the first hop?

1. Endstation (uses DNS stub resolver)
2. Recursive resolver (has root’s pubkey)
3. Referral to .edu
   + DS, RRSIG
4. Referral to upenn.edu
   + DS, RRSIG
5. Referral to upenn.edu
   + DS, RRSIG
6. Answer 1.2.3.4
   + RRSIG
7. Stub to Recursive Resolver channel
8. www.upenn.edu
   set DO bit

. (root) (-) root’s pubkey
.edu
.edu pubkey
upenn.edu
upenn pubkey
www.upenn.edu
(root’s pubkey)
DNS first hop protection

- Applications normally query a DNS stub resolver
- The stub resolver communicates over the network with a recursive resolver. How do we secure that path?
- Complex solutions exist (but rarely used)
  - e.g. employ a channel security mechanism between the stub and the validating recursive resolver:
    - TSIG, SIG(0), IPsec
- Run full-service validating resolver on endstation
- There may be other solutions, like DNScrypt – not standards based, only supported by a few resolvers, not widely used
- getdns can solve this problem
getdns: a new DNS library for applications

- getdns: A new application-friendly interface to the DNS
- Get and use arbitrary data in the DNS easily
- Get this data securely, authenticated with DNSSEC if it’s available
  - Full iterative resolver mode with validation
  - Validating stub resolver mode
- Designed by application developers. Most previous APIs have been developed by DNS protocol people with less concern for the needs of app developers.
getdns

- API specification:
  - Latest revision: February 2014
  - Creative Commons Attribution 3.0 Unported license

- An opensource implementation at [http://getdnsapi.net/](http://getdnsapi.net/)
  - A joint project of Verisign Labs and NLNet Labs
  - First release (0.1.0) in February 2014
  - Latest release (0.1.4) in August 2014
  - C library
  - Bindings in Python, and Node.js (upcoming: go, ruby, perl)
  - BSD 3 License
getdns features

- Asynchronous and synchronous modes of operation
- Sensible defaults suitable for consumption by most users
- But behavior highly configurable for users with advanced knowledge of the DNS
- DNS query results are returned in an easy to parse “response dictionary” data structure
- Members of the data structure can be lists, dictionaries, integers, and binary strings
- Can return DNSSEC status, and can be instructed to only return DNSSEC authenticated results
getdns functions

Four main functions defined.

getdns_address() Obtain IPv4 and/or IPv6 addresses
getdns_hostname() Obtain reverse DNS mappings
getdns_service() Obtain SRV record answers
getdns_general() General purpose DNS record query

Read the API specification for full details:

http://www.vpnc.org/getdns-api/
getdns response dictionary (partial)

{
    "answer_type": GETDNS_NAME_TYPE_DNS,
    "canonical_name": <bindata of "www.internet2.edu.">,
    "just_address_answers": [
        {
            "address_data": <bindata for 207.75.164.248>,
            "address_type": <bindata of "IPv4">
        },
        {
            "address_data": <bindata for 2001:48a8:68fe::248>,
            "address_type": <bindata of "IPv6">
        }
    ],
    "replies_full": [
        <bindata of 0x000081a00001000400000000103777777...>,
        <bindata of 0x000081a00001000400050000d03777777...>
    ], ...
}
getdns response dictionary (partial)

"dnssec_status": GETDNS_DNSSEC_SECURE,

"replies_tree":
[
    {
        "additional": [],
        "answer":
        [
            {
                "class": GETDNS_RRCLASS_IN,
                "name": <bindata for www.internet2.edu.>,
                "rdata":
                {
                    "cname": <bindata for webprod2.internet2.edu.>,
                    "rdata_raw": <bindata for webprod2.internet2.edu.>
                },
                "ttl": 120,
                "type": GETDNS_RRTYPE_CNAME
            },
            [...]
        ]
    }
]
getdns: example code: hostname lookup

# Example python code to query a domain name and
# return all associated IPv4 and IPv6 addresses.

hostname = sys.argv[1]

cxt = getdns.Context() 
extensions = {
    "return_both_v4_and_v6": getdns.GETDNS_EXTENSION_TRUE
}

results = cxt.address(name=hostname, extensions=extensions)
status = results['status']

if status == getdns.GETDNS_RESPSTATUS_GOOD:
    for addr in results['just_address_answers']:
        print addr['address_data']
else:
    print "%s: getdns.address() error: %d" % (hostname, status)

$ ./program.py www.internet2.edu
207.75.164.248
2001:48a8:68fe::248
getdns: example code: TLSA record lookup

# Example python code to lookup an authenticated TLSA # record for a domain name, transport, & service port.

qname = "_443._tcp.fedoraproject.org"
qtype = getdns.GETDNS_RRTYPE_TLSA

ctx = getdns.Context()
extensions = {
    "dnssec_return_only_secure": getdns.GETDNS_EXTENSION_TRUE
}

results = ctx.general(name=qname, request_type=qtype,
                      extensions=extensions)
status = results['status']

if status == getdns.GETDNS_RESPSTATUS_GOOD:
    # here we’d normally parse and do something useful with the
    # result data. For now just pretty print the dict.
    pprint.pprint(results)
else:
    print "%s: getdns.address() error: %d" % (hostname, status)
Questions or comments?
Shumon Huque
shuque @ verisign.com