UPPSALA
UNIVERSITET

Examensarbete 15 hp
Juni 2012

DANE with OpenSSL

PKIX certificate authentication through DNS using OpenSSL

Mathias Samuelson

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

DANE with OpenSSL

Mathias Samuelson

Background
X.509 is an ITU standard for a public key infrastructure (PKI), which specifies, among other things, formats for public key certificates, certificate requests, certificate revocation lists and certification path validation algorithm. The X.509 standard was primarily designed to support the X.500 structure. However, today's use cases centre mostly on the Internet.

IETF's Public-Key Infrastructure (X.509) working group has adapted the standard to the requirements and structure of the Internet. RFC 5280 specifies the PKIX Certificate and CRL Profile of the X.509v3 certificate standard.

PKIX certificates are used for validating the identity or identities of the communicating parties, and optionally establishing secure keying material for protection of a message or a communications channel. Authentication and establishment of a secure communications channel on top of TCP with the Transport Layer Security protocol (TLS, RFC 5247) or the Secure Sockets Layer protocol (SSL) is probably the most common application of PKIX on the Internet.

The IETF is converging on a standard for integration of X.509 Public Key Infrastructure with DNS and DNSSEC (DANE). In order to reach wide adoption, the concept must be validated through interoperability tests between multiple independent implementations.

Results
An implementation of the DANE standard has been demonstrated through an extension to the OpenSSL library. All use cases in the DANE standard has been validated to work as documented in the standard.

Conclusions
The DANE standard is implementable and reaches the results it sets out to achieve.

Handledare: Staffan Hagnell
Ämnesgranskare: Christian Rohner
Examinator: Olle Gällmo
IT 12 027
Tryckt av: Reprocentralen ITC

1. Introduction

This bachelor thesis is performed at .SE - Stiftelsen för Internetinfrastruktur and strives to develop a working demonstration of a DANE-enabled client using the OpenSSL library.

Background

X.509 is an ITU standard for a public key infrastructure (PKI), which specifies, among other things, formats for public key certificates, certificate requests, certificate revocation lists and certification path validation algorithm. The X.509 standard was primarily designed to support the X.500 structure. However, today's use cases centre mostly on the Internet. [1]

The Internet Engineering Task Force's (IETF) Public-Key Infrastructure (X.509) working group (PKIX WG) has adapted the standard to the requirements and structure of the Internet. Request For Comments (RFC) 5280 specifies the PKIX Certificate and CRL Profile of the X.509v3 certificate standard. [2][3]

PKIX certificates are used for validating the identity or identities of the communicating parties, and optionally establishing secure keying material for protection of a message or a communications channel. Authentication and establishment of a secure communications channel on top of TCP with the Transport Layer Security protocol (TLS, RFC 5247) [4] or the Secure Sockets Layer protocol (SSL) is probably the most common application of PKIX on the Internet. HTTP is the most prominent communications protocols taking advantage TLS/SSL. Other commonly used protocols are SMTP, IMAP, XMPP and SIP. X.509 Certificates is also used at other layers in the OSI model, e.g. IPSEC and for other purposes, e.g. Code signing.

Problem description

The IETF is converging on a standard for integration of X.509 Public Key Infrastructure with DNS and DNSSEC. In order to reach wide adoption, the concept must be validated through interoperability tests between multiple independent implementations.

This bachelor's thesis should assist the IETF effort with such interoperability testing by building an implementation according to the IETF standard.

Thesis objectives

The objective with this thesis is to:

- Make a proof of concept implementation with OpenSSL that validates the “DANE Protocol” currently in draft state within the IETF
- Survey the currently available proof of concept implementations and write a report

Delimitations

The DANE protocol was under development at the time of this thesis work with multiple drafts being published. In order to have a non-moving target, all analysis and work that went into this thesis is based on draft 12 of the DANE Internet Draft. This draft covered three different certificate usages, which were all implemented; subsequent drafts have introduced a fourth certificate usage, which was left unimplemented. [5]

2. Theory

Since the assertion embodied in the PKIX certification boils down to control over the domain name, and with the advent of DNSSEC, validation can be directly tied into DNS instead of a Certification Authority. This has a number of advantages; as opposed to using the long-lived assertions of a CA, the assertion published in DNS would be momentary, mitigating the certificate status checking vulnerabilities and increasing the service's responsiveness. Tying the validation into DNS can also preclude validation paths to the client's pre-installed database of CA's, in which all CA's may not be fully trusted. [6]

This move from using a local database of supposedly trusted third parties, into validating the service's certificate directly in the DNS, calls for a number of changes in how certificates are enrolled and validated. The following sections provide some background on the areas that will be affected by these changes.

X.509 Public Key Infrastructure (PKIX)

PKIX is the basis for most methods of securing communications channels between two parties that are unable or unwilling to agree on a shared cryptographic key ahead of time. With PKIX, each user generates a pair of keys that relates to each other such that one key can decrypt what the other encrypted and vice versa. One of these keys, called *public key,* is sent to a certification authority, which issues a signed certificate including the key; the user keeps the other key, called *private key*, private. [7]

When user A wants to prove its identity to user B it can do so by presenting its certificate, issued by certification authority C. If B trusts C, B trusts A. For B, C is referred to as a *trust anchor.*

An average user on the Internet is likely to come into contact with PKIX on a daily basis, with most of the complexity completely hidden by the applications used, such as the web browser. Specifically, the web browser manages the user's trust anchors.

Trust anchor management has the potential of introducing trust issues; if an application adds the certification authority D as a trust anchor to the user's trust anchor store, the user will now "trust" any other user A_1, A_2, and so on that can present a certificate that validates into D. If malicious user M finds a way to get D to issue a certificate for them, our user B will "trust" M as well.

This is also a problem for user A if malicious user M can impersonate A by getting D to issue a certificate A_M that validates into one of user B's trust anchors. This would allow M to impersonate A and for example act as a man in the middle between B and A. [8]

In 2011 this type of attack was demonstrated in the real world when a registration authority for Comodo [9] and the certification authority itself in the case of DigiNotar [10] were hacked on two separate occasions. This enabled the attacker to get the certificate authorities to issue valid certificates for domains not under the attacker's control. If the attacker can also subvert the user's connection, for example through DNS domain hijacking or simply by controlling the DNS resolver used by the user, the attacker can introduce himself as a man in the middle to collect intelligence, or attack the user financially, either directly (for example by gaining access to online banking credentials) or indirectly for example in click-fraud attacks. This is not to mention the overall privacy intrusion against the user.

The Domain Name System (DNS)

The DNS was introduced in the 1980-ies as a distributed host naming system that would scale better as the number of Internet connected hosts grew beyond what was practical to handle in a single host file that got FTP'd to all connected hosts regularly. Usually thought of as an upside-down tree with a root, denoted as the empty node (typically written out as '.'), it has an arbitrary number of branches on each level, referred to as "domains". [11][12]

A higher tier domain can delegate administrative ownership of a sub branch to another organization, a process referred to as "delegation of authority". The other organization

holds complete control over its sub branch, including having the privilege to delegate a sub branch of its sub branch to yet another organization.

If the domains are thought of as the DNS tree's branches, there are also leafs, referred to as "nodes" or "leaf nodes".

The DNS introduces resource records ("RR"), which can be of many different types to serve different purposes. A delegation of a sub domain to another organization is done by adding a leaf node to the parent domain that has the name of the delegated domain; an RR of type NS ("name server") is stored at this node, where the data consist of the name of the sub domain's DNS name servers. The sub domain completes the delegation by adding an RR of type SOA ("Start of Authority") at the top of the sub domain.

One particularly often used RR type is the A type ("address record"), which translates host names into IP addresses.

DNS-based Authentication of Named Entities (DANE)

The IETF, Internet Engineering Task Force, has been tasked to develop a protocol standard that addresses these problems. Like all other protocol work in the IETF, this is being done in a separate working group called DANE [5], where DANE stands for DNS-based Authentication of Named Entities.

With DANE, a domain owner can list the certificate authorities or end-entity certificates valid for PKIX-secured services under that domain, using the TLSA DNS record type introduced by this standard. This allows a DANE-capable client to ignore any trust assertions that are not listed for the domain it's connecting to. Effectively this establishes a direct trust relationship between the domain owner and the user, partially or completely removing all of the supposedly trusted third parties from the picture. [6]

When a domain owner publishes TLSA resource records (RR), the obvious intent is for the DANE to provide additional connection-establishment security, for example by mandating

what trust anchors the client uses for PKIX validation. For this to be realized the client must be able to *securely* find the correct TLSA RR for the server it's connecting to, which for all practical purposes requires end-to-end validation using DNS security extensions. [8]

The client usually relies on an external resolver, provided as a network service for example by the corporation that owns the computer, or by the Internet Service Provider if the computer is in a home. With DNS resolution performed external to the client, an argument can be made that a malicious third party can intercept the reply from the external resolver and change the content of the TLSA RR. This would effectively circumvent the additional security that all directly involved parties believed they had realized through the implementation of DANE.

The DANE standard requires the domain owner to publish TLSA RRs under *DNSSEC signed* zones only, and that the client that makes use of the TLSA RRs does so if and only if the TLSA RR validates correctly, and that the communication between client and validating resolver was secure. In order to meet the last requirement, the client can either use authenticated DNS queries and responses (for example TSIG or GSS-TSIG) [13][14] or it can use a secure network connection between itself and the validating resolver (for example using IPSec or other type of private network) [15]. A third alternative is for the client to perform DNSSEC validation itself.

DANE has broad-ranging applications to cover communications channel security, asynchronous message security, signaling security and so on. The scope for this thesis work has been communications channel security, specifically transport layer security, specifically using the OpenSSL library.

DNS Security Extensions (DNSSEC)

The DNS protocol was designed in the early 1980-ies, before most if not all of the well-known attacks against networked services and consequently the original design didn't include advanced security measures. A decade or more later this started to become an issue as malicious users on the Internet started to find ways to exploit DNS server weaknesses,

specifically to inject false information into the resolving name server's cache so that TCP/IP connections could be hijacked. This was for example demonstrated by Schuba in 1993. [16]

The IETF has attempted to add security as a transparent overlay to the basic DNS protocol, calling this the Domain Name System Security Extensions, DNSSEC for short. On a high level DNSSEC attempts to secure the name resolution process through the addition of a *chain of trust* where in the ideal world each set of DNS records is signed using a public-private key pair much like in PKIX. The resolving name server, now referred to as validating resolver, validates the signed record set against a known-good public key, or against a public key that validates against a chain into a known-good public key. [17]

The ideal known-good key is the one that signs the domain name system's top zone, the root zone, allowing validating resolvers to configure the public key part as a trusted anchor. Any record set signed by the root zone private key will be trusted by properly configured validating resolvers, including the DS records introduced by the DNS security extensions. The DS record makes an integrity and proof of origin assertion about a child domain's DNSKEY record. This DNSKEY, stored at the top of the child domain, signs, in other words makes an integrity and proof of origin assertion over, a second DNSKEY record. [18]

The DNSKEY records contains multiple pieces of data, for example what signing algorithm was used and the period of validity for the key; it also contains the actual public key for a key pair, where of course the private key is kept secret either in the name server configuration or offline. The first of the two signing key pairs, the one that had its integrity and proof of origin asserted by the parent domain's DS record is called key signing key, KSK for short. The second signing key pair, that had its integrity and proof of origin asserted by the KSK is called zone signing key, or ZSK. The ZSK is used for generating the signatures over all the remaining resource record sets in the zone. [18][19]

The separation of keys into KSK and ZSK is purely administrative and is primarily done so that the key used for signing the vast majority of data, the ZSK, can be trivially replaced without involving a third party, specifically the parent domain registrar. This is attractive

since the chance for a successful cryptanalysis to find the private key is proportional to the amount of data it has signed. In this regard, the ZSK is more exposed but can and should be replaced often.

This trust chain from the root zone of the domain name system all the way down to the leaf nodes enables, given that the chain of trust is intact, a domain owner to publish data in his or her domain that will be trusted from an integrity and proof of origin point of view. This specifically enables the domain owner to use the domain name system to publish PKIX certificate assertions either to constrain or add trust information for TLS validation by clients that supports DNS-based authentication of named entities (DANE).

OpenSSL

When the OpenSSL library is used for connection security, the application (for the purpose of this thesis "application" should in general be read as "client", as in the application that establishes the connection) uses a helper function that establishes a connection with the server and returns a BIO object. The BIO object is essentially a socket abstraction and can be used immediately for reading and writing data to the server. [7]

At this point the connection is not secure though. To secure it, the client and server must agree on capabilities along with the keying information. To achieve this, the application creates a SSL_CTX, "SSL context", object that holds the application's default connection configuration parameters.

Once the client has set its desired parameters on the SSL_CTX, it is used for creating an SSL object, which then gets connected to the server by associating it with the BIO object. The OpenSSL library now automatically negotiates a secure connection, typically using the server-side certificate only (the client can also offer a certificate for the user to authenticate its identity but in general this is not done).

Ironically, the one thing that the OpenSSL library *doesn't* do when it's initiating the secure connection is to actually validate the server certificate chain against a trust store [7]. To

add this crucial step the application must use the SSL_CTX_set_verify API call to add a verification callback function parameter to the SSL_CTX object. The callback function's role is to check the validation status of the server's certificate chain and return ok or not ok.

3. Method

To test the theory that DNS with DNSSEC can be used in the context of PKIX as described in the DANE IETF standard, a proof of concept was implemented using the following components:

- The OpenSSL library, which was extended with an implementation of the DANE concept.
- The libunbound validating client resolver library.
- A web server that hosts a number of sites via HTTP over TLS. The selected web server for this purpose was Apache 2.2.15.
- A DNS name server that hosts a domain with an A and a TLSA RR for each of the web sites. ISC BIND 9 was used.

The first two are client-side, while the web and DNS runs as a server, specifically on Centos 6 in VMware. All development work was performed on a Mac OS X 10.6 client, using gcc as a compiler.

The DANE standard has multiple use cases, each of which will be tested with a positive test, i.e. a test that validates that the client succeeds in establishing a TLS connection when the published TLSA RR matches the TLS certificate offered by the web server; and a negative test, i.e. a test that demonstrates that the client refuses to establish the connection with the web server when there is a mismatch. We will also demonstrate that a DNSSEC validation failure results in the client aborting the connection.

OpenSSL

This thesis work has made the following additions to the OpenSSL library:

1. An implementation of the DANE specification draft 12. This will be discussed in more detail below.

2. Changes to the s_client.c program that can be called as a component to the openssl(1) command line tool. These changes would be replicated in the application and in fact consist of a single call to the dane_verify function that's part of the DANE implementation.
3. A basic callback function that checks the server's certificate validation status.

Web server configuration

An Apache web server was installed and configured to provide https service for three simple sites under separate domain names. The web sites used certificates generated with OpenSSL, configured as a simple CA. Each site had its own certificate, with full certification paths up into the self-signed certification authority named "DANE CA". DNS records of type TLSA was created from each certificate and added to a zone file, which, in its signed version, was loaded by an ISC BIND server. The virtual hosts configurations are in appendix 1.

DNS infrastructure

An ISC BIND 9 name server was installed on the same host as the web server and had a signed copy of the example.com zone, and was configured to act as a full resolver for all other lookups. The computer where the DANE development and testing was taking place was configured to use the BIND server for name resolution so that the DANE-enabled client could connect to the local web server. The configuration and zone files (unsigned and signed) are in appendix 1.

Implementation of DANE

The DANE implementation consist of the single dane_verify() method that has the following prototype:

```
int dane_verify(SSL *con, char *s_host, short s_port);
```

The s_host and s_port are the server hostname and port that the application is connecting to. This method is called explicitly by the application that wants to use OpenSSL with this DANE extension.

An alternative implementation was also evaluated where the DANE specific work was included in the callback function, which automatically executed the DANE code. However, that approach was ultimately rejected for two reasons:

1. There can be only one verify callback function, and it was assumed that the application writer might have need to use his own function; with DANE evaluation being done in the callback, the application writer would effectively have been forced to edit the DANE implementation to include his own callback work, which is not ideal as this requires the application writer to understand the implementation of DANE rather than simply its interface.
2. The verify callback function interface doesn't include connection specific details; although it is possible through OpenSSL library calls to find first the SSL connection object, and then through calls to the socket API to find the connected port and the server IP address, which can usually be resolved into its hostname using the DNS, this approach was overly complex. Additionally, the DNS may resolve (correctly) the IP address into a hostname that is different from the one that the user connected to, which raises the question of who decides what hostname to use when looking up the DANE TLSA record in the DNS? For this thesis, it was decided that it should be the user's choice, in other words that the DANE implementation should take the hostname that the user has requested that the application connect to.

It would have been interesting to explore the callback function strategy in more detail, in particular if it was possible to register multiple functions that would be called in sequence, and ideally at more than a single point in the connection setup process. For example, there could be a callback before the certificate is subjected to PKIX validation and another callback that would execute after validation.

The dane_verify() method creates a validating DNS stub resolver using the libunbound library. A DNS lookup is made for a domain name on the form **_port._proto.hostname.domain**, for example _443._tcp.www.iis.se, query type TLSA. The

first label of this query name is the port, and is part of the method call; the second label is the protocol used for the connection, and is one of TCP, UDP, or SCTP. The protocol is found by the dane_verify() method through the socket API.

The DNS lookup could be invalid in either of two ways:

1. There was no data of type TLSA at the query name. In this case, the PKIX validation is allowed to proceed as if the application did not implement DANE at all.
2. There was data of type TLSA at the query name, but the DNSSEC validation failed. In this case the PKIX validation will be failed and the connection attempt should be rejected.

There is an additional edge case, where the validity status of the response was yet to be determined when this report was finalized for submission.

3. There was no response to the query for data of type TLSA at the query name. The IETF WG is discussing the best way to handle this situation [20] and an updated draft covering this case should be published shortly. Because it was still a very open question how to handle this case at the time of this writing, this error case has not been covered in this thesis.

If the DNS lookup returns a valid response, which means that there was data and that the data was correctly validating up into a DNSSEC trust anchor configured by the client, it is used either to put a constraint on the PKIX validation, or to set the PKIX trust anchor that should be used for validation. The exact mode of operation is decided by the data returned in the TLSA record, at the moment the following three modes of operation, or *usages*, are defined by DANE:

1. CA constraint, which specifies a CA certificate that must be found while performing PKIX validation over the server certificate;

2. Service certificate constraint, which specifies an end entity certificate that must be matched with the server certificate;
3. Domain-issued certificate, which specifies a certificate that must be used as trust anchor when performing PKIX validation over the server certificate.

All three usages are implemented for this thesis.

Additionally, the data returned in the TLSA record declares whether a full certificate is returned or if it's the result of a hash function over the certificate that's returned. For this thesis, only the full certificate case was implemented.

The source code for the proof of concept implementation is in appendix 2.

Test cases

To validate the assumption that DANE can be implemented with the OpenSSL library, we'll try each of the three usage cases with a positive and negative test case.

Usage 0 - CA constraint - Positive test

The DANE enabled client should pass PKIX validation and chain through a CA certificate that compares positive to the DER-encoded certificate bytes stored in the TLSA record.

```
openssl-dane msamuel$  ./s_client -CAfile cacert.pem -dane -connect www.0.0.0.example.com:443
CONNECTED(00000003)
DANE:www.0.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.0.0.0.example.com
DANE:dns name: _443._tcp.www.0.0.0.example.com
DANE DNS result is secure
DANE: Usage 0 Selector 0 Matching Type 0
DANE ca_constraint() chain of -1 length
DANE:www.0.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.0.0.0.example.com
DANE:dns name: _443._tcp.www.0.0.0.example.com
DANE DNS result is secure
DANE: Usage 0 Selector 0 Matching Type 0
DANE ca_constraint() chain of 3 length
DANE ca_constraint() cert 0 of 3.
DANE ca_constraint() certificates didn't match
DANE ca_constraint() cert 1 of 3.
DANE ca_constraint() certificates didn't match
```

```
DANE ca_constraint() cert 2 of 3.
DANE ca_constraint() certificates matches
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=DANE DNS/CN=www.0.0.0.example.com
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
 1 s:/C=US/ST=California/L=Mountain View/O=DANE DNS/CN=www.0.0.0.example.com
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
 2 s:/C=US/ST=CA/O=DANE CA/CN=DANE CA
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
---
Server certificate
< certificate and session details omitted >
Verify return code: 0 (ok)
---
openssl-dane msamuel$
```

Usage 0 - CA constraint - Negative test

For the CA constraint to fail we can replace the CA certificate the client loads from file (its "trust anchor" certificate) or we can change the DER-encoded certificate stored in the TLSA record. Easier still, we can simply change one of the bytes of the TLSA record, resign and reload the zone file.

```
openssl-dane msamuel$  ./s_client -CAfile cacert.pem -dane -connect www.0.0.0.example.com:443
CONNECTED(00000003)
DANE:www.0.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.0.0.0.example.com
DANE:dns name: _443._tcp.www.0.0.0.example.com
DANE DNS result is secure
DANE: Usage 0 Selector 0 Matching Type 0
DANE ca_constraint() chain of -1 length
DANE:www.0.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.0.0.0.example.com
DANE:dns name: _443._tcp.www.0.0.0.example.com
DANE DNS result is secure
DANE: Usage 0 Selector 0 Matching Type 0
DANE ca_constraint() chain of 3 length
DANE ca_constraint() cert 0 of 3.
DANE ca_constraint() certificates didn't match
DANE ca_constraint() cert 1 of 3.
DANE ca_constraint() certificates didn't match
DANE ca_constraint() cert 2 of 3.
DANE ca_constraint() certificates didn't match
DANE failed verification
openssl-dane msamuel$
```

Usage 1 - Certificate constraint - Positive test

The DANE-enabled client should pass PKIX validation and the end-entity certificate of the server should match the DER-encoded certificate in the TLSA record.

```
openssl-dane msamuel$ ./s_client -CAfile cacert.pem -dane -connect www.1.0.0.example.com:443
CONNECTED(00000003)
DANE:www.1.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.1.0.0.example.com
DANE:dns name: _443._tcp.www.1.0.0.example.com
DANE DNS result is secure
DANE: Usage 1 Selector 0 Matching Type 0
DANE:no peer certificate available
DANE: Passed validation for usage 1
DANE:www.1.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.1.0.0.example.com
DANE:dns name: _443._tcp.www.1.0.0.example.com
DANE DNS result is secure
DANE: Usage 1 Selector 0 Matching Type 0
DANE server_cert_constraint() certificates matches
DANE: Passed validation for usage 1
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=DANE DNS/CN=www.1.0.0.example.com
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
---
Server certificate
< certificate and session details omitted >
Verify return code: 0 (ok)
---
openssl-dane msamuel$
```

Usage 1 - Certificate constraint - Negative test

Again, by changing a single byte in the TLSA record we accomplish a negative test where the end-entity certificate of the server doesn't match the DER-encoded certificate in the TLSA record.

```
openssl-dane msamuel$ ./s_client -CAfile cacert.pem -dane -connect www.1.0.0.example.com:443
CONNECTED(00000003)
DANE:www.1.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.1.0.0.example.com
DANE:dns name: _443._tcp.www.1.0.0.example.com
DANE DNS result is secure
DANE: Usage 1 Selector 0 Matching Type 0
DANE:no peer certificate available
DANE: Passed validation for usage 1
DANE:www.1.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.1.0.0.example.com
DANE:dns name: _443._tcp.www.1.0.0.example.com
DANE DNS result is secure
DANE: Usage 1 Selector 0 Matching Type 0
DANE server_cert_constraint() certificates didn't match
DANE: Failed validation for usage 1
```

```
DANE failed verification
openssl-dane msamuel$
```

Usage 2 - Domain-issued certificate - Positive test

This usage of DANE enables a domain owner to store a DER-encoded CA certificate in the TLSA record so that a DANE-enabled client can load it as a trust anchor. Note that the CAfile option to the s_client program has been omitted.

```
openssl-dane msamuel$ ./s_client -dane -connect www.2.0.0.example.com:443
CONNECTED(00000003)
DANE:www.2.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.2.0.0.example.com
DANE:dns name: _443._tcp.www.2.0.0.example.com
DANE DNS result is secure
DANE: Usage 2 Selector 0 Matching Type 0
DANE:www.2.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.2.0.0.example.com
DANE:dns name: _443._tcp.www.2.0.0.example.com
DANE DNS result is secure
DANE: Usage 2 Selector 0 Matching Type 0
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=DANE Web server/CN=www.2.0.0.example.com
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
 1 s:/C=US/ST=California/L=Mountain View/O=DANE Web server/CN=www.2.0.0.example.com
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
 2 s:/C=US/ST=CA/O=DANE CA/CN=DANE CA
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
---
Server certificate
< certificate and session details omitted >
Verify return code: 0 (ok)
---
openssl-dane msamuel$
```

Usage 2 - Domain-issued certificate - Negative test

By commenting out the TLSA record from the zone file we see that the PKIX validation now fails because there is no CA certificate in the certificate path trusted by the client.

```
openssl-dane msamuel$ ./s_client -dane -connect www.2.0.0.example.com:443
CONNECTED(00000003)
DANE:www.2.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.2.0.0.example.com
DANE:dns name: _443._tcp.www.2.0.0.example.com
DANE DNS result is secure
dane_verify_callback error with cert at depth: 1
dane_verify_callback issuer  = /C=US/ST=CA/O=DANE CA/CN=DANE CA
dane_verify_callback subject = /C=US/ST=CA/O=DANE CA/CN=DANE CA
dane_verify_callback error 19:self signed certificate in certificate chain
```

```
DANE:www.2.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.2.0.0.example.com
DANE:dns name: _443._tcp.www.2.0.0.example.com
DANE DNS result is secure
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=DANE Web server/CN=www.2.0.0.example.com
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
 1 s:/C=US/ST=California/L=Mountain View/O=DANE Web server/CN=www.2.0.0.example.com
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
 2 s:/C=US/ST=CA/O=DANE CA/CN=DANE CA
   i:/C=US/ST=CA/O=DANE CA/CN=DANE CA
---
Server certificate
< certificate and session details omitted >
Verify return code: 19 (self signed certificate in certificate chain)
openssl-dane msamuel$
```

DNSSEC validation failure

If the client fails to validate the DNSSEC signatures for the TLSA record, the client *must* fail and abort the connection. A DNSSEC validation can fail in many different ways, either because of administrative errors, for example that the RR signatures have expired, or due to a man in the middle attack where the signed RR has been changed, for example by a malicious or hacked recursive resolver, so that the RR and the signatures generated over the RR doesn't match. From the point of view of the validating client or resolver, all these errors are equivalent so we only need to test for a single type of error, for example expired RR signatures.

```
openssl-dane msamuel$ ./s_client -dane -connect www.0.0.0.example.com:443
CONNECTED(00000003)
DANE:www.0.0.0.example.com:443
DANE synthesize_tlsa_domain() dns name: _443._tcp.www.0.0.0.example.com
DANE:dns name: _443._tcp.www.0.0.0.example.com
DANE DNS result is bogus: validation failure <_443._tcp.www.0.0.0.example.com. TYPE65534 IN>:
signature expired from 172.16.52.182 for trust anchor example.com. while building chain of trust
DANE failed verification
```

These tests demonstrate that the DANE protocol can be implemented using the OpenSSL protocol.

4. Survey

At the moment the available implementations of DANE are relatively limited and mostly consist of research projects and products of thesis works like this one.

Research Project 1: Implementing a DANE validator

This research project by Pieter Lewis, University of Amsterdam [21], asks the question:

"Is DANE in its current form implementable and does it achieve its goal of securely binding DNS names to TLS certificates on end-hosts?"

The project looks at various combinations of purchased and self-signed certificates, and after having published TLSA records for each combination, the project then validates the combination under a tool written in Python called *swede* that "can create and verify TLSA records".

The project concludes that the DANE standard is implementable:

"Apart from the issues arising from the lack of clarity of the phrase "pass PKIX validation" combined with usage 2, DANE is a specification that can be implemented and could be the killer application DNSSEC needs for wide-spread deployment."

DANE implementation for NSS

Matt McCutchen has written a patch [22] to NSS, which offers the promise of extending the Mozilla class of browers to support DANE. The web site doesn't provide much detail beyond the actual patch though.

5. Conclusions and suggested future work

The goal for this thesis was to demonstrate that it is possible to implement DANE within the framework of OpenSSL and this goal has been achieved with the proof of concept implementation that is also made freely available on the Internet:

https://github.com/mathiassamuelson/openssl-dane

DANE seems to solve a couple of interesting problems. First of all it enables a domain owner to signal to a PKIX client what Certification authority certificates should be considered valid; this has the huge benefit that the domain owner takes control over the client's trust anchor, specifically eliminating any potential rogue CA certificates in the client's trust anchor repository. As demonstrated above, the usages referred to as "CA constraint" and "Certificate constraint", does enable the domain owner to gain this control over the client's trust anchor.

When DANE is implemented by the domain order as a means of restricting what Certificate Authorities are valid for TLS services in that domain, it's effectively signaling to the client that it should not validate up into any other CA. As such it certainly *improves* the security from the user's point of view as it makes a man in the middle attack significantly less likely to succeed.

Secondly, it also provides a means for a domain owner to introduce their own CA certificate through which the clients shall perform PKIX validation. This completely changes the economy of the use of PKIX as the domain owners can issue their own end-entity certificates largely at zero cost. That this works was also demonstrated in the test cases for the "Domain-issued certificate" usage above.

It can be argued that this usage of DANE also has a positive effect on the security from the user's point of view, as it eliminates the "This is an untrusted certificate, do you want to

trust it anyway?" browser warning. In this case, implementation of DANE enables the domain owner to signal to the browser that this is indeed a trusted certificate, which eliminates the false positive browser warning.

The area that DANE does not attempt to address is certificate management through DNS for the client side. DANE for the client side seem, at least on the surface, to hold significant potential for improving authentication security for access to sensitive information or for connecting to secure network ingress points (such as VPN or shell access). It would be interesting to see some future work in that area.

Another area that also needs future work is for the integration of this implementation into some application that is actually being used by real users, for example a web browser, SIP client, or some library that is used by many other applications, for example libcurl.

Ultimately, DANE could be included with APIs that are an integral part of the operating system, either as part of OpenSSL and similar libraries, or as a wrapper API that allows an application programmer to incorporate DANE validation at the lowest possible effort and cost.

There is also a need for tools that can help to operationalize DANE, for example tools that can take a certificate, in file format or through an application-level connection over TLS, and generates the corresponding TLSA record.

As a final point - .SE - Stiftelsen för Internetinfrastruktur hosted a DANE seminar in Stockholm in early January where Staffan Hagnell and Jakob Schlyter introduced .SE's interest in this area, with a general introduction to DANE. The two theses projects that's been supervised by .SE during the fall of 2011 and spring of 2012, including this one, was then presented by the respective student. The presentations were well received by an audience of about 25 people.

6. References

[1] Recommendation X.509 (11/08) [Internet]. International Telecommunications Union; 2008 [Updated 2011 Feb, 2012 Apr; cited 2012 Jun 26]. Available from http://www.itu.int/rec/T-REC-X.509-200811-I

[2] Public-Key Infrastructure (X.509) (pkix) [Internet]. Internet Engineering Task Force. [Cited 2012 Jun 26]. Available from http://datatracker.ietf.org/wg/pkix/charter/

[3] Cooper D, Santesson S, Farrell S, Boeyen S, Housley R, Polk W. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [Internet]. Internet Engineering Task Force. 2008 May. [Cited 2012 Jun 26]. Available from http://www.ietf.org/rfc/rfc5280.txt

[4] Dierks T, Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.2 [Internet]. Internet Engineering Task Force. 2008 Aug. [Cited 2012 Jun 26]. Available from http://tools.ietf.org/html/rfc5246

[5] DNS-based Authentication of Named Entities (dane). Internet Engineering Task Force. [Cited 2012 Jun 26]. Available from http://datatracker.ietf.org/wg/dane/charter/

[6] Using Secure DNS to Associate Certificates with Domain Names For TLS. Internet Engineering Task Force; 2011 Sep [Cited 2012 Jun 26]. Available from http://tools.ietf.org/id/draft-ietf-dane-protocol-12.txt

[7] Viega J, Messier M, Chandra P. Network Security with OpenSSL. Sebastopol, California, United States of America. O'Reilly. 2002.

[8] Barnes R. Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE). Internet Engineering Task Force. 2011 Oct. [Cited 2012 Jun 26]. Available from http://www.rfc-editor.org/rfc/rfc6394.txt

[9] Prince B. Comodo Attack Sparks SSL Certificate Security Discussions [Internet]. CRN. 2011 Mar 24. [Cited 2012 Jun 26]. Available from http://www.crn.com/news/security/229400284/comodo-attack-sparks-ssl-certificate-security-discussions.htm

[10] Prins J.R. Interim Report - DigiNotar Certificate Authority breach “Operation Black Tulip” [Internet]. Fox-IT BV. 2011 Sep 5. [Cited 2012 Jun 26]. Available from http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2011/09/05/diginotar-public-report-version-1/rapport-fox-it-operation-black-tulip-v1-0.pdf

[11] Mockapetris P. DOMAIN NAMES - CONCEPTS AND FACILITIES [Internet]. Internet Engineering Task Force. 1987 Nov. [Cited 2012 Jun 26]. Available from http://www.ietf.org/rfc/rfc1034.txt

[12] Mockapetris P. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION [Internet]. Internet Engineering Task Force. 1987 Nov. [Cited 2012 Jun 26]. Available from http://www.ietf.org/rfc/rfc1035.txt

[13] Vixie P, Gudmundsson O, Eastlake 3rd D, Wellington, B. Secret Key Transaction Authentication for DNS (TSIG) [Internet]. Internet Engineering Task Force. 2000 May. [Cited 2012 Jun 26]. Available from http://www.ietf.org/rfc/rfc2845.txt

[14] Kwan S, Garg P, Gilroy J, Esibov L, Westhead J, Hall R. Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG) [Internet]. Internet Engineering Task Force. 2003 Oct. [Cited 2012 Jun 26]. Available from http://www.ietf.org/rfc/rfc3645.txt

[15] Kent S, Seo K. Security Architecture for the Internet Protocol [Internet]. Internet Engineering Task Force. 2005 Dec. [Cited 2012 Jun 26]. Available from http://tools.ietf.org/html/rfc4301

[16] Schuba CL. Addressing Weaknesses in the Domain Name System Protocol [MSc thesis]. W Lafayette, IN, USA: Purdue University. 1993 Aug.

[17] Arends R, Austein R, Larson M, Massey D, Rose S. DNS Security Introduction and Requirements [Internet]. Internet Engineering Task Force. 2005 Mar. [Cited 2012 Jun 26]. Available from http://tools.ietf.org/html/rfc4033

[18] Arends R, Austein R, Larson M, Massey D, Rose S. Resource Records for the DNS Security Extensions [Internet]. Internet Engineering Task Force. 2005 Mar. [Cited 2012 Jun 26]. Available from http://tools.ietf.org/html/rfc4034

[19] Arends R, Austein R, Larson M, Massey D, Rose S. Protocol Modifications for the DNS Security Extensions [Internet]. Internet Engineering Task Force. 2005 Mar. [Cited 2012 Jun 26]. Available from http://tools.ietf.org/html/rfc4035

[20] Rescorla E. Behavior in the face of no answer? [Internet]. Internet Engineering Task Force. 2012 May 3. [Cited 2012 Jun 26]. Available from http://www.ietf.org/mail-archive/web/dane/current/msg04845.html

[21] Lexis P. Research Project 1: Implementing a DANE validator [Internet]. Amsterdam, Netherlands: University of Amsterdam, System and Network Engineering. 2012 Feb 9. [Cited 2012 Jun 26]. Available from http://staff.science.uva.nl/~delaat/rp/2011-2012/p29/report.pdf

[22] McCutchen M. Cryptographic service identity [Internet]. 2011 Apr 13. [Cited 2012 Jun 26]. Available from https://mattmccutchen.net/cryptid/index.html

Appendix 1 - Configurations

Apache web server configuration

```
<VirtualHost 172.16.52.182:443>
	SSLEngine on
	SSLCertificateChainFile /etc/pki/tls/certs/www.0.0.0.example.com-combined.crt
	SSLCertificateFile /etc/pki/tls/certs/www.0.0.0.example.com.crt
	SSLCertificateKeyFile /etc/pki/tls/private/www.0.0.0.example.com.key
	<Directory /var/www/vhosts/example.com/httpsdocs>
	AllowOverride All
	</Directory>
	DocumentRoot /var/www/vhosts/example.com/httpsdocs
	ServerName www.0.0.0.example.com
</VirtualHost>
<VirtualHost 172.16.52.183:443>
	SSLEngine on
	SSLCertificateFile /etc/pki/tls/certs/www.1.0.0.example.com.crt
	SSLCertificateKeyFile /etc/pki/tls/private/www.1.0.0.example.com.key
	<Directory /var/www/vhosts/example.com/httpsdocs>
	AllowOverride All
	</Directory>
	DocumentRoot /var/www/vhosts/example.com/httpsdocs
	ServerName www.1.0.0.example.com
</VirtualHost>
<VirtualHost 172.16.52.184:443>
	SSLEngine on
	SSLCertificateChainFile /etc/pki/tls/certs/www.2.0.0.example.com-combined.crt
	SSLCertificateFile /etc/pki/tls/certs/www.2.0.0.example.com.crt
	SSLCertificateKeyFile /etc/pki/tls/private/www.2.0.0.example.com.key
	<Directory /var/www/vhosts/example.com/httpsdocs>
	AllowOverride All
	</Directory>
	DocumentRoot /var/www/vhosts/example.com/httpsdocs
	ServerName www.2.0.0.example.com
</VirtualHost>
```

BIND DNS server configuration and DNS data

```
=== /etc/named.conf
//
// named.conf
//
// Provided by Red Hat bind package to configure the ISC BIND named(8) DNS
// server as a caching only nameserver (as a localhost DNS resolver only).
//
// See /usr/share/doc/bind*/sample/ for example named configuration files.
//

options {
	listen-on port 53 { 127.0.0.1; 172.16.52.182; };
```

```
        listen-on-v6 port 53 { ::1; };
        directory       "/var/named";
        dump-file       "/var/named/data/cache_dump.db";
        statistics-file "/var/named/data/named_stats.txt";
        memstatistics-file "/var/named/data/named_mem_stats.txt";
        allow-query     { any; };
        recursion yes;

        dnssec-enable yes;
        dnssec-validation yes;
        dnssec-lookaside auto;

        /* Path to ISC DLV key */
        bindkeys-file "/etc/named.iscdlv.key";
};

logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};

zone "example.com" IN {
        type master;
        file "example.com.signed";
};
zone "52.16.172.in-addr.arpa" IN {
        type master;
        file "52.16.172.in-addr.arpa";
};


=== /var/named/example.com
$TTL 1D
$ORIGIN example.com.
@       IN SOA  master rname.invalid. (
                                        2       ; serial
                                        1D      ; refresh
                                        1H      ; retry
                                        1W      ; expire
                                        3H )    ; minimum
        NS      master
master  A       172.16.52.182
www     A       172.16.52.182
www.0.0.0       A       172.16.52.182
_443._tcp.www.0.0.0.example.com. IN TYPE65534 \# 674
0000003082029b30820204a003020102020900d3a8d831c43133f8300d06092a864886f70d01010505003
03e310b3009060355040613025553310b3009060355040813024341311030 0e060355040a130744414e4
52043413110300e060355040313074441 4e45204341301e170d313131323039303435383139 5a170d313
431323038303435383139 5a303e310b3009060355040613025553310b3009060355040813024341311030
00e060355040a130744414e452043413110300e060355040313074441 4e4520434130819f300d06092a8
64886f70d010101050003818d00308189028181009 5e4e73274e8ceeb6e55a31d9168ca68cb7a1cc5f453
cf484d2c0b622df3c10d490bb1accbd032b272e804b0ae01039d1e8abe9b6b2b7d37871d24fc5831129581
2e9b10db94aaa23412132681e272422593a747eaa0d5283b484f596dbb045a657a5f574e0451a03fb9492
e7c82b5594248b68b262df489ca31a6c00febb8150203010001a381a030819d301d0603551d0e04160414
```

7dd98c0984817361cc886a9e9848c888cc5ec3e0306e0603551d230467306580147dd98c0984817361cc8
86a9e9848c888cc5ec3e0a142a440303e310b30090603550406130255533l0b30090603550408130243 4
13110300e060355040a130744414e452043413110300e06035504031307444l4e45204341820900d3a8d
831c43133f8300c0603551d13040530030101ff300d06092a864886f70d0101050500038181003ab1e8dc8
83f62814f4510aad39f7efee53d452b3490a8ad89de21e152c130bb960011e9bd85568aa2d52e0ded3eb2b
65fcc9ba996d3835ff86b24b79454b4f151ee44355cb459ad13728567443e6d282832dff91f321fd9f718cbc2
7d1b3ccd366e0bae50c0c9b89662e200be334ac31b97b5b510fc16c23f6a2c2a6deb4fb8
www.1.0.0 A 172.16.52.183
_443._tcp.www.1.0.0.example.com. IN TYPE65534 \# 683
010000308202a43082020da003020102020900d3a8d831c43133fa300d06092a864886f70d01010505003
03e310b30090603550406130255533l0b30090603550408130243413110300e060355040a130744414e4
52043413110300e06035504031307444l4e45204341301e170d3131313232363130353232335a170d313
231323235313035323233 5a306d310b30090603550406130255533113301106035504081 30a43616c696
66f726e6961131630140603550407130d4d6f756e7461696e20566965773111300f060355040a13084441
4e4520444e53311e301c06035504031315777777 2e312e302e302e6578616d706c652e636f6d30819f300
d06092a864886f70d010101050003818d00308189028181 00d1a0382b00b50b3e3fbed8ce45db8e7d4929
4bd3b4164d905ee510d66e36685e4e216e3ccd8176f1c2aaf8f6263e9f93924e9edb9a46eb3b58eb1264ad
eae749eaa276a416da074a3aace8d62f850c188547e89de2371203f241a4e6a7b95302e728215b86c75e8
8f314d407227283fd7ff9bc4ec9c5573f00b013bd9149173f0203010001a37b307930090603551d13040230
00302c06096086480186f842010d041f161d4f70656e53534c2047656e6572617465642043657274696 6
963617465301d0603551d0e0416041467f6c62d1b50d6801d9ee72939f7d1dbc98416fb301f0603551d230
418301680147dd98c0984817361cc886a9e9848c888cc5ec3e0300d06092a864886f70d01010505000381
810055d0b3e25f175ac67487f067d20dbd7fad459e14f4c8e4ef351fbea560647fa7d18b7f5b510a8e82a1a4
b192a9d2ea4157f403792546a3eb00f23945d4b64460b8a064c4f9b75544a557df029276b25d71c3d8f2e7
3ff28732a36a17d7f38bcc45d100aa5775154aa9a3b66b4a8202801af8747bb31901d74a10af1aa995ffe3
www.2.0.0 A 172.16.52.184
_443._tcp.www.2.0.0.example.com. IN TYPE65534 \# 674
0200003082029b30820204a003020102020900d3a8d831c43133f8300d06092a864886f70d01010505003
03e310b30090603550406130255533l0b30090603550408130243413110300e060355040a130744414e4
52043413110300e06035504031307444l4e45204341301e170d3131313230393034353831395a170d313
431323038303453831395a303e310b30090603550406130255533l0b30090603550408130243413110 3
00e060355040a130744414e452043413110300e06035504031307444l4e4520434130819f300d06092a8
64886f70d010101050003818d00308189028181 0095e4e73274e8ceeb6e55a31d9168ca68cb7a1cc5f453
cf484d2c0b622df3c10d490bb1accbd032b272e804b0ae01039d1e8abe9b6b2b7d37871d24fc5831129581
2e9b10db94aaa23412132681e272422593a747eaa0d5283b484f596dbb045a657a5f574e0451a03fb9492
e7c82b5594248b68b262df489ca31a6c00febb8150203010001a381a030819d301d0603551d0e04160414
7dd98c0984817361cc886a9e9848c888cc5ec3e0306e0603551d230467306580147dd98c0984817361cc8
86a9e9848c888cc5ec3e0a142a440303e310b30090603550406130255533l0b30090603550408130243 4
13110300e060355040a130744414e452043413110300e06035504031307444l4e45204341820900d3a8d
831c43133f8300c0603551d13040530030101ff300d06092a864886f70d0101050500038181003ab1e8dc8
83f62814f4510aad39f7efee53d452b3490a8ad89de21e152c130bb960011e9bd85568aa2d52e0ded3eb2b
65fcc9ba996d3835ff86b24b79454b4f151ee44355cb459ad13728567443e6d282832dff91f321fd9f718cbc2
7d1b3ccd366e0bae50c0c9b89662e200be334ac31b97b5b510fc16c23f6a2c2a6deb4fb8

```
=== /var/named/example.com.signed
; File written on Fri Jan 13 22:28:09 2012
; dnssec_signzone version 9.7.0-P2-RedHat-9.7.0-5.P2.el6_0.1
example.com.            86400   IN SOA  master.example.com. rname.invalid. (
                                2         ; serial
                                86400     ; refresh (1 day)
                                3600      ; retry (1 hour)
                                604800    ; expire (1 week)
                                10800     ; minimum (3 hours)
                                )
                        86400   RRSIG   SOA 5 2 86400 20120213052809 (
```

```
                20120114052809 5307 example.com.
                RzxfT538gf8haJytKRo7llwQVuetmdEB0xhx
                4DShFN/3LzsDrCy7Yeurvcml4xxsujoAteet
                23Hp6LhIFp9TmxXlb6yRhErnDtmcopXULGkk
                VH4+TBjFlfTE4Y73qckf )
        86400   NS      master.example.com.
        86400   RRSIG   NS 5 2 86400 20120213052809 (
                20120114052809 5307 example.com.
                d/Vv1TtzbLdBvhnwfzCS8dYEDsMCPDSXunVe
                5l5QaiqAuLdbqmhNjJirTvswoT06/TMQmGuX
                gclgs1QVcOCWrJn91OztsaDFO2AwG4BMJ83n
                sG89P0HeQ8lUt0rs4Peh )
        10800   NSEC    www.0.0.0.example.com. NS SOA RRSIG NSEC DNSKEY
        10800   RRSIG   NSEC 5 2 10800 20120213052809 (
                20120114052809 5307 example.com.
                OX9bUx+QLq0s8C0SPNmI/xLZ5au3D4IiuEUJ
                jhIVVMMPxKLZN/vrLgNU1FoAw24TZc3rX16l
                9W/M2Ipw4sHlKiG1CawFdqPAj39Fbv0LbaTC
                TcH5pY/7qCJSEdG7vL53 )
        86400   DNSKEY  256 3 5 (
                AwEAAa8gZwUFWpjsVAdoRmbazvMkoP3XMUnQ
                oyvfREwrIuEv7bBxaLSoaL7lAhVOC78b/boJ
                R+fFk9qIyPEewIOWjrBN9bqnok5EjYrZNx6W
                gQHJg86zKWbf6QadqAs/rrFjgQ==
                ) ; key id = 5307
        86400   DNSKEY  257 3 5 (
                AwEAAeMXCMn8LYkfmedk4SvIaKFbfFPEwG33
                YUnWgp8cRUe4o8BnWXEAFxwFzxdYLnZNiN3p
                YCDE5GdFtWMhoJUXETNN98J9v3gurC+cgvZX
                5FVutPwa6trPmMOo/wcs5uEG1GormFslWpD0
                NDLEwsSaZuzrBlP07mbuV/LtiVWWYglfog/9
                v8vCVBiC19PnnwqvHR01dEEswHXWsuHiWXaa
                ahCurWlvmf+i4F/4IhslPJRgfvzflvi921JM
                jvwnTE1zS2NAAfIghQgdgwPrhAKTj4b5DhUe
                Lg/DscYDfDusOpL1gnyDwPgpr/dA8Wsgf1lp
                gVvKPJ5y1atOO3P6IZ5wYvs=
                ) ; key id = 9868
        86400   RRSIG   DNSKEY 5 2 86400 20120213052809 (
                20120114052809 5307 example.com.
                obzaYsH4bWD6NEYlHuhxO8dMIGhlY1sOdn8w
                4FW6oxOzIJQ/iEEPK1itneyZMlnVlCs7Ptv3
                k23Jc9ATy5urJmX4S5uC1AN65ncXTxlb+fn+
                VvWR7XtyaVzZgXKkLAj4 )
        86400   RRSIG   DNSKEY 5 2 86400 20120213052809 (
                20120114052809 9868 example.com.
                x0tkmaZsuDux1AcS99j9FdQIMjwg1zMEx/85
                p8GulTSzIkgkQn5ht0YAzQb36iU2eB4dFI6+
                tmXouDIIF+kw8rnpQMRAAtqy1W4aFj7qCoe7
                tynZ2wkJgzFsvFoH8382gklRRwNejdCpYwhf
                +kZd3Qv9fbKEQRaMRnhOIFZbjeBHbClRD+ps
                ZyTOfixGkkvuIgL5Of4a1gCc+J0lSdKjMONC
                7PzP+cV8YIBYPfl34HdydgByEaLpfCYn3wIu
                PUw9eeNs3TNG3Gg3W7CfntQcrlaIv54YBcdJ
                zb8wNoUZ5alHvJDfQKzkgSSqcDCNqOFXrKa7
                N6S8DUPLQbT9vaiM2w== )
www.0.0.0.example.com.  86400   IN A    172.16.52.182
```

```
                86400   RRSIG   A 5 6 86400 20120213052809 (
                          20120114052809 5307 example.com.
                          WJKiwP9PAXePhl1/i5R+sWXptPTGXv6bfUgR
                          hFO1hEdmQrLNe6hrVNiZMPS3dQINvBe1PHsJ
                          xC5Ze4imEHHPCwakOOfHX7g9al2YKQAynpV5
                          iiIyXLDr58jlGJr8adKd )
                10800   NSEC    _443._tcp.www.0.0.0.example.com. A RRSIG NSEC
                10800   RRSIG   NSEC 5 6 10800 20120213052809 (
                          20120114052809 5307 example.com.
                          ppyGZWzzD9XIM3DDJ+LSx4j0zdWVY0MnS1+e
                          yKF0d35Nbg6J6QSzRsd7EImBhF6r6jLGlKiD
                          oBks48fD/g1BZZx0nlefFfxF86MfGNjIn7lH
                          o7hD5mgCb81dJTfXXYwY )
_443._tcp.www.0.0.0.example.com. 10800 IN NSEC www.1.0.0.example.com. RRSIG NSEC
TYPE65534
                10800   RRSIG   NSEC 5 8 10800 20120213052809 (
                          20120114052809 5307 example.com.
                          nzqHAvxD9ej+7tukXGIMvRkdzjK3XQ7UwOdP
                          YCdssRtuI0PLOVRD1+4JacELrfeGrfjUsfCk
                          7AvIvLFYdw+6+3H6XbUfelmqjpqxvoQQcVFR
                          PpX7KahBfRVkuoGRvorI )
                86400   TYPE65534 \# 674 ( 0000003082029B30820204A0030201020209
                          00D3A8D831C43133F8300D06092A864886F7
                          0D0101050500303E310B3009060355040613
                          025553310B300906035504081302434131 10
                          300E060355040A130744414E452043413110
                          300E0603550403130744414E45204341301E
                          170D313131323039303435383139 5A170D31
                          343132303830343538313 95A303E310B3009
                          0603550406130255533 10B30090603550408
                          130243413110300E060355040A130744414E
                          452043413110300E0603550403130744414E
                          4520434130819F300D06092A864886F70D01
                          0101050003818D0030818902818100 95E4E7
                          3274E8CEEB6E55A31D9168CA68CB7A1CC5F4
                          53CF484D2C0B622DF3C10D490BB1ACCBD032
                          B272E804B0AE01039D1E8ABE9B6B2B7D3787
                          1D24FC58311295812E9B10DB94AAA2341213
                          2681E272422593A747EAA0D5283B484F596D
                          BB045A657A5F574E0451A03FB9492E7C82B5
                          594248B68B262DF489CA31A6C00FEBB81502
                          03010001A381A030819D301D0603551D0E04
                          1604147DD98C0984817361CC886A9E9848C8
                          88CC5EC3E0306E0603551D23046730658014
                          7DD98C0984817361CC886A9E9848C888CC5E
                          C3E0A142A440303E310B3009060355040613
                          025553310B300906035504081302434131 10
                          300E060355040A130744414E452043413110
                          300E0603550403130744414E452043418209
                          00D3A8D831C43133F8300C0603551D130405
                          30030101FF300D06092A864886F70D010105
                          0500038181003AB1E8DC883F62814F4510AA
                          D39F7EFEE53D452B3490A8AD89DE21E152C1
                          30BB960011E9BD85568AA2D52E0DED3EB2B6
                          5FCC9BA996D3835FF86B24B79454B4F151EE
                          44355CB459AD13728567443E6D282832DFF9
```

```
                        1F321FD9F718CBC27D1B3CCD366E0BAE50C0
                        C9B89662E200BE334AC31B97B5B510FC16C2
                        3F6A2C2A6DEB4FB8 )
                86400   RRSIG   TYPE65534 5 8 86400 20120213052809 (
                        20120114052809 5307 example.com.
                        Sehbx0kjt/SK7ISHRTCrsDALnRPJMBt3x7Y4
                        P9Jb7dALIZUQvj51VlfCqvoycKGRcOYqjy8n
                        L9sClkbtpuMyBbh6yavarKr8IAnh4YQ6/Rs6
                        SDUjMGwxFqd/yEBLDS+V )
www.1.0.0.example.com.  86400   IN A    172.16.52.183
                86400   RRSIG   A 5 6 86400 20120213052809 (
                        20120114052809 5307 example.com.
                        PkIdulf/v5Gasljq5vE1G3lVIzc0nRglyDtu
                        +sk20BoWPIkgRf3IPsnT/fwltQza8yaNVTg7
                        LAQceXupOEkqJZCW4UuRW/t6AacaVcxNrGtg
                        fkDWKubPR70zvD/5EtfB )
                10800   NSEC    _443._tcp.www.1.0.0.example.com. A RRSIG NSEC
                10800   RRSIG   NSEC 5 6 10800 20120213052809 (
                        20120114052809 5307 example.com.
                        RUR0BOk6q4d2neaWtJPAAax6WQcHiwvm0f/x
                        iI8r4cXXhrhYpPZM1wZsC/dBJo1AG9Jr2Spi
                        zFYG2gW7EiXiiuzLgiCLIEzhPqhxNThvWf3b
                        yDiO6R5Aqzjz4hPmXdCU )
www.2.0.0.example.com.  86400   IN A    172.16.52.184
                86400   RRSIG   A 5 6 86400 20120213052809 (
                        20120114052809 5307 example.com.
                        T/MK6JjpUBTNc7D+roYq54Fk6Mu2vNjsX9Dg
                        vny0dE0oDmvxKqcSS/XN3RMMahAGPRbDOHIP
                        aTvs0OUj2l44WKnaKF0Kcqp2kcHKxT5C/Ey7
                        g584nxvi4hcQkLoR8tHe )
                10800   NSEC    _443._tcp.www.2.0.0.example.com. A RRSIG NSEC
                10800   RRSIG   NSEC 5 6 10800 20120213052809 (
                        20120114052809 5307 example.com.
                        qhO7ULtaQnQ0O/Kta1zg/k+0QfdVCCbYwFeW
                        gdG1fzLZvWDXsPGrSIYIYDYDeJZ06q+z0bGI
                        2w5OcLWMYymxGTIXOuYITXf1GMvJYD9epXeV
                        tRPzWcNeAFPI/Viqds6w )
_443._tcp.www.2.0.0.example.com. 10800 IN NSEC master.example.com. RRSIG NSEC TYPE65534
                10800   RRSIG   NSEC 5 8 10800 20120213052809 (
                        20120114052809 5307 example.com.
                        Hi874Ymv4KpffKAmGWMYYRWhWFjWx5RiRUt/
                        pjPwD0Tx0IJjagqc10TvP4xcWYdAgWX17EVO
                        Cdl+XGtXzSsN37QX8G56tjBPA55D1D+ruKpw
                        bZ2IYnxpxRtGknyICqXC )
                86400   TYPE65534 \# 674 ( 0200003082029B30820204A0030201020209
                        00D3A8D831C43133F8300D06092A864886F7
                        0D0101050500303E310B3009060355040613
                        025553310B300906035504081302434131 10
                        300E060355040A130744414E452043413110
                        300E0603550403130744414E45204341301E
                        170D313131323039303435383139 5A170D31
                        343132303830343435383139 5A303E310B3009
                        0603550406130255533 10B30090603550408
                        130243413110300E060355040A130744414E
                        452043413110300E0603550403130744414E
                        4520434130819F300D06092A864886F70D01
```

```
                                0101050003818D003081890281810095E4E7
                                3274E8CEEB6E55A31D9168CA68CB7A1CC5F4
                                53CF484D2C0B622DF3C10D490BB1ACCBD032
                                B272E804B0AE01039D1E8ABE9B6B2B7D3787
                                1D24FC58311295812E9B10DB94AAA2341213
                                2681E272422593A747EAA0D5283B484F596D
                                BB045A657A5F574E0451A03FB9492E7C82B5
                                594248B68B262DF489CA31A6C00FEBB81502
                                03010001A381A030819D301D0603551D0E04
                                1604147DD98C0984817361CC886A9E9848C8
                                88CC5EC3E0306E0603551D23046730658014
                                7DD98C0984817361CC886A9E9848C888CC5E
                                C3E0A142A440303E310B3009060355040613
                                025553310B300906035504081302434131 10
                                300E060355040A130744414E452043413110
                                300E06035504031307 44414E452043418209
                                00D3A8D831C43133F8300C0603551D130405
                                30030101FF300D06092A864886F70D010105
                                0500038181003AB1E8DC883F62814F4510AA
                                D39F7EFEE53D452B3490A8AD89DE21E152C1
                                30BB960011E9BD85568AA2D52E0DED3EB2B6
                                5FCC9BA996D3835FF86B24B79454B4F151EE
                                44355CB459AD13728567443E6D282832DFF9
                                1F321FD9F718CBC27D1B3CCD366E0BAE50C0
                                C9B89662E200BE334AC31B97B5B510FC16C2
                                3F6A2C2A6DEB4FB8 )
                        86400   RRSIG   TYPE65534 5 8 86400 20120213052809 (
                                20120114052809 5307 example.com.
                                YUPwOJBaGlllEqs/4TFCs/XLAczlbHsB/7IV
                                ZXDFqjcBWjFJRPQS7+k28cp5U2Stc39MsTUn
                                djLzwuH5UyDvr8wWv0/OT1pKAW+ORVK7Cu47
                                st5wsTDoiwrJFeLO74sd )
master.example.com.     86400   IN A    172.16.52.182
                        86400   RRSIG   A 5 3 86400 20120213052809 (
                                20120114052809 5307 example.com.
                                kSp8OPrMx7q2MQ5radOoMU6QQ7/7zSJkwycc
                                lpngnqX1h89f7Cg0+c/8ZMHS8A84AbGe2Gx4
                                BswtlhJvHZ1CJLLbMdAHgTWnp1AgfdASXVlj
                                MqjD90T3uVlulW3H4TyU )
                        10800   NSEC    www.example.com. A RRSIG NSEC
                        10800   RRSIG   NSEC 5 3 10800 20120213052809 (
                                20120114052809 5307 example.com.
                                SmxgywkP2CV660J4EVrrsLL7yV+atvKFKf7N
                                /x3Dlagc062hdFyi+Jvbmd5d7OtH137WJ103
                                4ggvI+R7lAiSPdaHBQK15mgKPrF/8HiF0WEY
                                yh4G8fky2oRp7H6J+hBw )
www.example.com.        86400   IN A    172.16.52.182
                        86400   RRSIG   A 5 3 86400 20120213052809 (
                                20120114052809 5307 example.com.
                                hEyHUSGSiI9O9dnz+tQZphWiVx8hha0pxN4p
                                jP2BOLIOcaY7iYso2JTJlJXSxhkQZ6XMRmDG
                                UeMI5Ne7Cm/wih+QXY3xwbM2viSD0L0HDUkS
                                eMhQOEPWznmDNOWdq+uA )
                        10800   NSEC    example.com. A RRSIG NSEC
                        10800   RRSIG   NSEC 5 3 10800 20120213052809 (
                                20120114052809 5307 example.com.
```

```
                        GXf+A4KhXe83AN2rRVGB1g4v938xlrqG/Q0J
                        P1/JaJjq+g4JvU4X4/gipC2blvjjMA5vKx3b
                        MYgHXcCyewlkuniswgcV+1RJry6+dxzeyvfx
                        5gW6o7L/ElOrkbVMsrKc )
_443._tcp.www.1.0.0.example.com. 10800 IN NSEC www.2.0.0.example.com. RRSIG NSEC
TYPE65534
                10800   RRSIG   NSEC 5 8 10800 20120213052809 (
                        20120114052809 5307 example.com.
                        lw9PkbRtQ2VMWDwTrQUpBxf0UjMZUGQwnirn
                        10apHKJX/X1P9XQ/hcst8Fe8aWQTX11p5/9q
                        bGEDFruw30K10rog/BlD4xVMo6OMrnBWZdTW
                        yQxO2cRFIV0QxLxspgf9 )
                86400   TYPE65534 \# 683 ( 010000308202A43082020DA0030201020209
                        00D3A8D831C43133FA300D06092A864886F7
                        0D0101050500303E310B3009060355040613
                        025553310B300906035504081302434131 10
                        300E060355040A130744414E452043413110
                        300E0603550403130744414E45204341301E
                        170D313131323236313035323233 5A170D31
                        323132323531303532323335A306D310B3009
                        060355040613025553311330110603550408
                        130A43616C69666F726E6961131630140603
                        550407130D4D6F756E7461696E2056696577
                        3111300F060355040A130844414E4520444E
                        53311E301C060355040313157777772E312E
                        302E302E6578616D706C652E636F6D30819F
                        300D06092A864886F70D010101050003818D
                        0030818902818100D1A0382B00B50B3E3FBE
                        D8CE45DB8E7D49294BD3B4164D905EE510D6
                        6E36685E4E216E3CCD8176F1C2AAF8F6263E
                        9F93924E9EDB9A46EB3B58EB1264ADEAE749
                        EAA276A416DA074A3AACE8D62F850C188547
                        E89DE2371203F241A4E6A7B95302E728215B
                        86C75E88F314D407227283FD7FF9BC4EC9C5
                        573F00B013BD9149173F0203010001A37B30
                        7930090603551D1304023000302C06096086
                        480186F842010D041F161D4F70656E53534C
                        2047656E657261746564204365727469666 9
                        63617465301D0603551D0E0416041467F6C6
                        2D1B50D6801D9EE72939F7D1DBC98416FB30
                        1F0603551D230418301680147DD98C098481
                        7361CC886A9E9848C888CC5EC3E0300D0609
                        2A864886F70D01010505000381810055D0B3
                        E25F175AC67487F067D20DBD7FAD459E14F4
                        C8E4EF351FBEA560647FA7D18B7F5B510A8E
                        82A1A4B192A9D2EA4157F403792546A3EB00
                        F23945D4B64460B8A064C4F9B75544A557DF
                        029276B25D71C3D8F2E73FF28732A36A17D7
                        F38BCC45D100AA5775154AA9A3B66B4A8202
                        801AF8747BB31901D74A10AF1AA995FFE3 )
                86400   RRSIG   TYPE65534 5 8 86400 20120213052809 (
                        20120114052809 5307 example.com.
                        Hp3N6lF9RjfEFhV4UFeb6gu31it3Mvy3Sj9/
                        lelannlFgIP+9Nji+KZANaOc3WXtbDPmzC4x
                        RP96WT9IAVF+VnUlJLnnVZSXgK5WEKekZBzz
                        DJ3XdbOztBkfWYKZBmLL )
```

Appendix 2 - Source Code

```
/*
 * Copyright I 2012 Mathias Samuelson. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
 * IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
 * IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */

#include <stdio.h>
#include <errno.h>
#include <arpa/inet.h>
#include <unbound.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <openssl/bio.h>
#include <openssl/x509.h>
#include <openssl/x509v3.h>

#include “dane.h”

BIO *b_err;
int get_tlsa(struct ub_result *result, char *s_host, short s_port);
int ca_constraint(const SSL *con, const X509 *tlsa_cert, int usage);
int service_cert_constraint(const X509 *con_cert, const X509 *tlsa_cert);
int synthesize_tlsa_domain(char *tlsa_domain, const SSL *con, char *hostname);

int synthesize_tlsa_domain(char *tlsa_domain, const SSL *con, char *hostname) {
        int peerfd;
        peerfd = SSL_get_fd(con);
        socklen_t len;
```

```
        struct sockaddr_storage addr;
        char ipstr[INET6_ADDRSTRLEN];
        char node[NI_MAXHOST];
        char dns_name[256];
        char proto[4];
        int sock_type, optlen;
        int port;
        int retval;
        len = sizeof addr;
        getpeername(peerfd, (struct sockaddr*)&addr, &len);
        // deal with both Ipv4 and Ipv6:
        if (addr.ss_family == AF_INET) {
           struct sockaddr_in *s = (struct sockaddr_in *)&addr;
           port = ntohs(s->sin_port);
           inet_ntop(AF_INET, &s->sin_addr, ipstr, sizeof ipstr);
              struct sockaddr_in sa;
              sa.sin_family = AF_INET;
              inet_pton(AF_INET, ipstr, &sa.sin_addr);
              int res = getnameinfo((struct sockaddr*)&sa, sizeof(sa), node, sizeof(node), NULL, 0, 0);
        } else { // AF_INET6
              struct sockaddr_in6 *s = (struct sockaddr_in6 *)&addr;
           port = ntohs(s->sin6_port);
           inet_ntop(AF_INET6, &s->sin6_addr, ipstr, sizeof ipstr);
        }
        optlen = sizeof(sock_type);
        retval = getsockopt(peerfd, SOL_SOCKET, SO_TYPE, &sock_type, &optlen);
        if (retval == -1) {
              BIO_printf(b_err, "synthesize_tlsa_domain failed to get socket type: %d\n", retval);
              return -1;
        }
        switch (sock_type) {
              case SOCK_STREAM:
                    sprintf(proto, "tcp");
                    break;
              case SOCK_DGRAM:
                    sprintf(proto, "udp");
                    break;
        }

        retval = sprintf(tlsa_domain, "_%d._%s.%s", port, proto, node);
        if(retval < 0) {
              printf("failure to create dns name\n");
              return -1;
        }
        BIO_printf(b_err,"DANE synthesize_tlsa_domain() dns name: %s\n", tlsa_domain);

        return 0;
}

int dane_verify_callback(int ok, X509_STORE_CTX *store) {
        if (b_err == NULL)
              b_err=BIO_new_fp(stderr,BIO_NOCLOSE);

        char data[256];
        if (! ok) {
              X509 *cert = X509_STORE_CTX_get_current_cert(store);
```

```
            int depth = X509_STORE_CTX_get_error_depth(store);
            int err = X509_STORE_CTX_get_error(store);

            BIO_printf(b_err, "dane_verify_callback error with cert at depth: %d\n", depth);
            X509_NAME_oneline(X509_get_issuer_name(cert), data, sizeof(data));
            BIO_printf(b_err, "dane_verify_callback issuer  = %s\n", data);
            X509_NAME_oneline(X509_get_subject_name(cert), data, sizeof(data));
            BIO_printf(b_err, "dane_verify_callback subject = %s\n", data);
            BIO_printf(b_err, "dane_verify_callback error %i:%s\n", err,
                  X509_verify_cert_error_string(err));
      }

      return ok;
}
int dane_verify(SSL *con, char *s_host, short s_port) {
      struct ub_result *dns_result;
      struct ub_ctx* ctx;
      char dns_name[256];
      int retval;

      if (b_err == NULL)
            b_err=BIO_new_fp(stderr,BIO_NOCLOSE);
      BIO_printf(b_err, "DANE:%s:%d\n", s_host, s_port);

      ctx = ub_ctx_create();
      if(!ctx) {
            printf("error: could not create unbound context\n");
            return -1;
      }
      if( (retval=ub_ctx_resolvconf(ctx, "/etc/resolv.conf")) != 0) {
            printf("error reading resolv.conf: %s. errno says: %s\n",
                  ub_strerror(retval), strerror(errno));
            return -1;
      }
      if( (retval=ub_ctx_hosts(ctx, "/etc/hosts")) != 0) {
            printf("error reading hosts: %s. errno says: %s\n",
                  ub_strerror(retval), strerror(errno));
            return -1;
      }
      if( (retval=ub_ctx_add_ta_file(ctx, "keys")) != 0) {
            printf("error adding keys: %s\n", ub_strerror(retval));
            return 1;
      }

      synthesize_tlsa_domain(dns_name, con, s_host);
      BIO_printf(b_err,"DANE:dns name: %s\n", dns_name);
      retval = ub_resolve(ctx, dns_name, 65534, 1, &dns_result);
      if(retval != 0) {
            printf("resolve error: %s\n", ub_strerror(retval));
            return -1;
      }
      if (dns_result->secure)
            BIO_printf(b_err, "DANE DNS result is secure\n");
      else if (dns_result->bogus) {
            BIO_printf(b_err, "DANE DNS result is bogus: %s\n", dns_result->why_bogus);
            SSL_shutdown(con);
```

```
        return -1;
    } else {
        // should allow PKIX validation to proceed but without DANE
        BIO_printf(b_err, "DANE DNS result is insecure\n");
        return -1;
    }

    if(dns_result->havedata) {
        int I;
        for (I = 0; dns_result->data[i] != NULL; i++) {
            unsigned char usage, selector, matching_type;
            unsigned char *tlsa_bytes;

            if (dns_result->len[i] < 35) {
                // must have at least 1+1+1+32 bytes for the SHA-256 case
                BIO_printf(b_err, "DANE: Not enough data: %d available\n",
                    dns_result->len[i]);
                return -1;
            }
            unsigned char *rdata = (unsigned char *)dns_result->data[i];
            usage = (char) *rdata++;
            selector = (char) *rdata++;
            matching_type = (char) *rdata++;
            tlsa_bytes = (unsigned char *)rdata;
            X509 *tlsa_cert;
            tlsa_cert = d2i_X509(NULL, &tlsa_bytes, dns_result->len[i]-3);

            BIO_printf(b_err, "DANE: Usage %d Selector %d Matching Type %d\n",
                usage, selector, matching_type);

            if (selector != 0)
                continue;
            if (matching_type != 0)
                continue;

            int retval;
            switch (usage) {
                case 0:
                    return ca_constraint(con, tlsa_cert, usage);
                    //break;
                case 1: {
                    X509 *cert = NULL;
                    cert = SSL_get_peer_certificate(con);
                    retval = service_cert_constraint(cert, tlsa_cert);
                    if (retval == 0)
                        BIO_printf(b_err, "DANE: Passed validation for usage 1\n");
                    else
                        BIO_printf(b_err, "DANE: Failed validation for usage 1\n");
                    X509_free(cert);
                    return retval;
                    break;
                }
                case 2: {
                    SSL_CTX *con_ctx = SSL_get_SSL_CTX(con);
                    X509_STORE *vfy_store;
```

```
                        if (!(vfy_store = X509_STORE_new())) {
                            BIO_printf(b_err, "DANE dane_verify error creating store");
                            retval = -1;
                        } else {
                            X509_STORE_add_cert(vfy_store, tlsa_cert);
                            SSL_CTX_set_cert_store(con_ctx, vfy_store);
                            retval = 0;
                            break;
                        }
                    }
                }
                X509_free(tlsa_cert);
            }
        } else
            return 0;

        (void)BIO_flush(b_err);
        return retval;
}

/*
 *      Returns:        0 if successfully matching certificate to TLSA record bytes
 *                      -1 if there was no match
 */
int ca_constraint(const SSL *con, const X509 *tlsa_cert, int usage) {
        STACK_OF(X509) *cert_chain = NULL;
        cert_chain = SSL_get_peer_cert_chain(con);
        BIO_printf(b_err, "DANE ca_constraint() chain of %d length\n",
            sk_X509_num(cert_chain));
        int ret_val;
        ret_val = 0;

        if (cert_chain != NULL) {
            int I;
            for (I = 0; I < sk_X509_num(cert_chain); i++) {
                BIO_printf(b_err, "DANE ca_constraint() cert %d of %d.\n",
                    I, sk_X509_num(cert_chain));

                if (X509_cmp(tlsa_cert, sk_X509_value(cert_chain, i)) < 0) {
                    ret_val = -1;
                    BIO_printf(b_err, "DANE ca_constraint() certificates didn't match\n");
                } else {
                    BIO_printf(b_err, "DANE ca_constraint() certificates matches\n");
                    return 0;
                }
            }
        }
        return ret_val;
}

/*
 *      Returns:        0 if successfully matching certificate to TLSA record bytes
 *                      -1 if there was no match
 */
//int service_cert_constraint(const SSL *con, const unsigned char *tlsa_bytes, int tlsa_len) {
```

```
int service_cert_constraint(const X509 *con_cert, const X509 *tlsa_cert) {
	int ret_val;
	ret_val = 0;

	if (con_cert != NULL) {
		if (X509_cmp(tlsa_cert, con_cert) != 0) {
			ret_val = -1;
			BIO_printf(b_err, “DANE server_cert_constraint() certificates didn’t match\n”);
		} else {
			BIO_printf(b_err, “DANE server_cert_constraint() certificates matches\n”);
			// Must return immediately in case there are non-matching certificates
			return 0;
		}
	} else
		BIO_printf(b_err,”DANE:no peer certificate available\n”);

	return ret_val;
}
```