University of Amsterdam
System & Network Engineering

Research Project 1

Extended Validation using DNSSEC

Authors:
Danny Groenewegen
danny.groenewegen@os3.nl

Pieter Lange
pieter.lange@os3.nl

Coordinators:
Michiel Leenaars
michiel@nlnet.nl

Rick van Rein
rick@openfortress.nl

Abstract

Remote trust on the web is mostly handled by so called Certificate Authorities. companies, government bodies or other types of organisations that users go to to obtain their own certificates. There is a significant leap of faith involved: why should you blindly trust the hundreds of Certificate Authorities preloaded in your browser to not abuse their root certificates, when many Certificate Authorities are organisations you don't know anything about - which means you might not want to trust them for all purposes, certainly not if you can avoid it. What if the websites and services you care about can publish the certificates they use safely and authoritatively through the DNS? Historically, the answer was that DNS itself was not safe enough. With DNSSEC you get a chain of trust from the signed root of the internet to the service you want to connect to.
We have researched different ways of doing this and made available an add-on to the new Firefox 4.0 browser software which enables end-users and server administrators to leverage the DNSSEC chain of trust as anchor for their certificates.

February 7, 2011

Acknowledgements

We would like to thank the following people and organisations for their guidance and support during our project:

- Michiel Leenaars (NLnet Foundation) for his supervision and incredible support;
- Rick van Rein (OpenFortress) for his insight, ideas and feedback;
- The System and Network Engineering (University of Amsterdam) group for the means and opportunity to conduct the research project.

Contents

Chapter 1

Introduction

> Anyone who spends any time with DNSSEC realizes eventually it'll be used for much more than simply validating IP addresses.
>
> *Dan Kaminsky, 17-10-2009*

~

The Domain Name System Security Extensions, or 'DNSSEC', are a suite of specifications that secure the Domain Name System (DNS). PKIX (Public Key Infrastructure X.509) is a set of specifications used for – among other things – the SSL certificates in HTTPS.

This report assumes a basic understanding of DNS(SEC), X.509 and public key infrastructures in general, however we'll try to give a brief overview of the most important features used in our implementation.

DNSSEC

DNSSEC was designed to cope with the security issues introduced by the legacy DNS specification. The original system was specified[16] in 1987 and was not designed to deal with the hostile nature of the Internet as we know it now. Security researchers have proven[1] for many years that the DNS system is vulnerable to a number of issues such as (but not limited to):

1. Cache poisoning due to predictable transaction IDs[10] or name-chaining;
2. Man-in-the-middle and other packet interception attacks;
3. "Betrayal" by trusted server (for example OpenDNS[21]).

DNSSEC is a system that enables every zone operator to cryptographically sign the records in their zone and have the *parent zone* provide a *signed delegation* along with its regular `NS` delegation. This creates a chain of trust that end users can verify by only having a copy of the public key of the root zone. (Refer to page 18 for a simplified view of the chain of trust.)

Because of this chain of trust the DNSSEC infrastructure as specified in RFC4033 through RFC4035 provides end users with origin authenticity and data integrity.

Quick overview of DNSSEC chain of trust validation

DNSSEC validators (i.e. libraries and validating resolvers[14]) will validate records *bottom-up*. Every DNSSEC enabled answer is accompanied by digital signatures in the `RRSIG` record (*resource record signature*). The digital signature can be verified by locating the correct public key in the `DNSKEY` record in the zone its *apex*. `DNSKEY` records are authenticated by the `DS` (*delegation signer*) record in the parent zone.

In practice the `DNSKEY` records are usually split in a *key signing key* (KSK) and a *zone signing key* (ZSK), introducing another step in the validation process. The parent zone's `DS` record will associate with the KSK, which in turn signs the ZSK. The ZSK can be used to sign the other records in the zone and be changed more often.

The `RRSIG` records have a limited period in which they are valid which is independent from the original 'Time To Live'. This means a zone must be periodically resigned to assure validity of the answers.

An example signature 'chase' validating the entire DNSSEC chain of trust from the DNS label `www.os3sec.org` up onto the trusted (root) anchor can be read in appendix B.

X.509 Certificates

X.509 is a standard for public key infrastructures, single sign-on and privilege management infrastructures. There are multiple versions of the X.509 standard, which originally only supported a hierarchical system of *certificate authorities* for signing certificates. For the purposes of this project we limited ourselves to the *public key infrastructure* and *public key certificate* format specifications of the standard as specified by the *PKIX* working group of the IETF[2].

As with DNSSEC there are chains of trust to be followed up onto a *trusted anchor*. Certificate authorities issue certificates that are bound to a distinguished name such as a domain name or an e-mail address. The certificate chain of trust has to be provided by the peer or otherwise publicly available. Some semantics are similar, but there are significant differences with regard to DNSSEC:

1. There are *multiple root anchors* preconfigured in most implementations;
2. Certificates contain an extendible set of *policies* or *constraints*;
    - Certificates have a specific *purpose* (i.e. certificates can or cannot sign other certificates);
    - New constraints can be added, clients that do not support them cannot continue;
3. Compromised certificates have to be *revoked* using a certificate revocation list (CRL) or the online certificate status protocol (OCSP).

Some of these differences are clear when looking at the certificate structure:

- Certificate
    - Version
    - Serial number
    - Algorithm ID
    - Issuer
    - Validity

        * Not before
        * Not after
    - Subject
    - Subject public key information
        * *Public key algorithm*
        * *Public key*
    - Extensions
- Certificate signature algorithm
- Certificate signature

The problem with X.509

Because most client applications have a big set[3] of root anchors defined in agreements between the web browser manufacturers and certificate authorities[4], clients are unable to make a final judgement on the authenticity of a certificate. A content provider might choose to use one certificate authority, but if the client receives a valid certificate signed by another certificate authority, the client will accept it without a problem.

This enables foreign goverments or anyone else with leverage over these certificate authorities to do hostile takeovers of websites and other services secured with these certificates.

What DNSSEC enables us to do

Because the *trust* of DNSSEC is rooted in one single *trust anchor* a lot of effort has been put in to ensure that people will keep trusting the DNS root. The impartiality of the root is of upmost importance. Which is why the private key for the root was generated in a special ceremony with trusted representatives from Internet Society chapters and other impartial observers from all over the world.

These representatives have signed the public key on their own, seeding the trust for these keys. By utilizing the PGP web of trust we're able to verify the authenticity of the root – that is, if we trust the representatives.

Security researchers have noted that this root anchor and subsequent chains will receive a lot more scrutiny because of their importance. This means we will have the ability to verify the authenticity of any answer, so long as we trust the entities in the chain – just by having the public key part of the DNS root.

So the concept is clear; the hard question is *how* we will do this. Design decisions now will have a lot of impact in the future.

Overview

The next chapter will discuss two different answers to *how* we should solve this problem, after which we'll discuss our implementation and finally we'll discuss our findings.

Chapter 2

Standards

The nice thing about standards is that there are so many of them to choose from.

Andrew Tanenbaum

~

Now that the DNS root servers have been successfully signed since the 15^{th} of July 2010, efforts are underway to make use of the *chain of trust* DNSSEC provides to bootstrap trust into other protocols. This chapter discusses the design parameters and where choices are to be made. We also discuss two methods that are currently in development: the Internet Engineering Task Force (IETF) draft standard and the method used by Dan Kaminsky's tool *Phreebird*.

2.1 Design Considerations

There are many ways one can associate a DNS entry with a x509 certificate. It is possible to put a copy of the entire certificate into a record on the same label to create a one-to-one relation with the certificate. The facility to publish a wide array of different certificate types in DNS is already implemented in the `CERT`[7] record type, but it is important to note that the RFC doesn't specify any requirements for publishing the `CERT` record at related labels.

One also has to consider the size of such records: certificates can easily span multiple kilobytes, while a hash of that certificate will certainly be adequate to use as a base for building a relation with the certificate. However, it is wise to keep the option of having full certificates open; clients might not implement all hash functions and having a full copy of the certificate would require no more steps than a simple comparison operation.

2.1.1 Record Type

The new standard could extend an existing resource record type or specify an entirely new resource record type. Both options have merit.

A new record type would mean that we do not have to deal (as much) with backward compatibility, as old clients will not requests these records and therefore do not deal with our policies – these clients will have to validate the certificate using some other method to bootstrap their trust.

Reusing or overloading an old resource record type has the benefit of less (re-)integration work for the numerous DNS management tools. The difficulty is to find a type that will conform to the data constraints set by the new standard. Special care has to be taken *not* to simply put the

data in a `TXT` record. This method is tempting because `TXT` record types allow you to put in any character strings, which allow you to make key=value associations within the `TXT` record.[19]

However, another much more recent RFC notes that using `TXT` for these purposes is almost always a bad idea. The most important objections come from the fact that it is hard to coordinate the use of such a free namespace. There are no semantics to prevent collisions. Also, the amount of `TXT` records will increase. Multiple records could be associated with one label while each record has its own purpose. This in turn will increase the size of DNS answers, forcing more connections to TCP.

To make the parsing of the record more easy for the client the record should also include a separate parameter indicating the hash algorithm, if used.

2.1.2 Policy and Legacy

Arguably the most important part of the standard is creating proper recommendations for client policies, keeping support for the current x509 public key infrastructure in mind. A standard that doesn't provide backward compatibility with legacy systems will prove to be useless because lack of deployment.

To provide legacy support we'll also need to make policy decisions in the standard, because the new bootstrap system and the old certificate policies have conflicting interests. Certificates have constraints like the *common name* of the server. Certificates *expire*, usually after a year or so. Certificates can sometimes sign other certificates – surely we do not want to install new root certificates in our client application, so we'll have to strip certain data from the certificate before we'll add it into our certificate cache.

Should the client honor the restrictions in the certificate, even if it got a valid DNSSEC-authenticated response? After all, DNS(SEC) also specifies these fields:

- *common name* relates to the fully qualified domain name;
- DNS records expire all the time due to TTL and expiring RRsig records.

There is also the issue of dealing with multiple TLS enabled services running on one host, possibly each with their own certificate and key material. This would mean that there are multiple records containing certificate data at the same label. Should the record also include the port number associated with the certificate in the record, or should the client just accept any record that matches?

2.1.3 Placement in DNS Hierarchy

Again we are faced with multiple options. The most obvious choice is to put the certificate payload – whether it is a hash or the full certificate – in a special or adopted record type on the label of the host we're trying to access. This provides the benefit of transparent `CNAME` redirection to the final DNS label with the `A` or `AAAA` record containing the IP address of the host.

Of course the label itself can also contain the certificate payload - if it is a hash with sufficient compression to fit inside a DNS label (256 characters). This means the client would have to first connect to the service to receive the certificate, after which the client can do a lookup for the DNS entry containing the hash prepended to the original label.

2.2 IETF

The Internet Engineering Task Force (IETF) has set up a working group to deal with the issue of "Using Secure DNS to Associate Certificates with Domain Names For TLS". IETF working groups

are organized in areas depending on the topics discussed. Current areas include: Applications, General, Internet, Operations and Management, Real-time Applications and Infrastructure, Routing, Security and Transport. The *DNS-based Authentication of Named Entities* (dane) working group is operating in the 'security area'.

2.2.1 Dane Working Group

The working groups' objective is specified as follows[5]:

> "Specify mechanisms and techniques that allow Internet applications to establish cryptographically secured communications by using information distributed through DNSSEC for discovering and authenticating public keys which are associated with a service located at a domain name."

As with any other working group, the standard will be distilled through a process of getting 'rough consensus' on specific issues in the specification. Sometimes the process is also influenced by 'running code'; if working implementations exist they can be studied 'in the wild' and be evaluated. Working groups that discuss topics which affect a majority of users or administrators are naturally more difficult to achieve full consensus on all matters. A mailing list is used for communication between anyone willing to participate in the standards process.

The working group has the following milestones:

- Apr 2011 ⇒ First WG draft of standards-track protocol for using DNS to associate hosts with keys for TLS and DTLS
- May 2011 ⇒ First WG draft of standards-track protocols for using DNS to associate hosts with IPsec
- Sep 2011 ⇒ Protocol for using DNS to associate domain names with keys for TLS and DTLS to IESG
- Sep 2011 ⇒ Protocols for using DNS to associate domain names with keys for IPsec to IESG
- Nov 2011 ⇒ Recharter

With revision three of the document describing (D)TLS association in DNSSEC released the working group is well on schedule to meet the milestones.

2.2.2 Current Specification

Please note that this subsection describes a document that is currently being discussed by the working group. Details may change in the future or already have changed. Refer to the original document for implementation details.
The current draft document[18] deals only with TLS and DTLS server identification. It is further limited to PKIX certificates as specified by RFC5280[2].

Certificate associations are made using either a fingerprint of the certificate or by using the full binary certificate itself and a domain name. This is done using a newly specified DNS record type, `TLSA`. The specification currently expects the `TLSA` record at the same DNS label.

The format of the `TLSA` specifies three parameters:

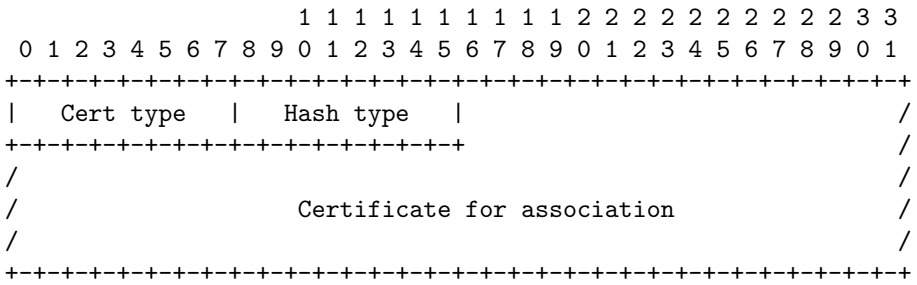
1. Certificate type [one octet];
2. Hashing algorithm, if used [one octet];

3. Certificate for association.

Four certificate types are supported. A DNS administrator can choose to either select the certificate of the TLS server itself (end-entity) in hashed form or in full, or to associate the DNS label with a parent certificate, also hashed or in full, to be provided in the certificate-chain by the TLS server. The first type of association (end-entity certificate) is more useful for smaller to medium sites with just one server per label. The second type is useful for larger (i.e. load-balancing) setups where every single server gets its own certificate which is signed by the parent certificate.

The hash type is `0` when no hashing is used (certificate type 2 and 4), otherwise the correct value most be picked from a to-be-assigned IANA registry. Currently only SHA1 (1), SHA256(2) and SHA384(3) are supported.

Lastly, the certificate for association is given. This will be either the hash or the full DER encoded certificate. This results in the following wire format for the `TLSA` record:



The document further describes a small set of policies regarding the use of `TLSA` records as trusted anchors. Care must be taken that the record is obtained using a trusted source; either a local (on the same machine) validating resolver, a local library or a secured channel to a remote validating name server. If the client fails to associate the certificate using `TLSA`, it must try to validate it in the normal fashion (using certificate authorities pre-configured in the client). If a client fails to associate an end-entity certificate it must fail with an *access_denied* error.

2.2.3 Discussion Topics

Some important implementation parameters are left out of the document because no consensus has been formed on these matters yet. For instance "Issue 1" has not reached consensus: How to deal with multiple TLS-based services under one domain name?

Multiple TLS services on one DNS label Right now it is impossible to know what port and/or protocol a `TLSA` record belongs to. There are basically two main categories for solutions:

1. Include the port number and protocol in the `TLSA` record response;
2. Include the port number and protocol in the request.

Solution #1 will simply add another parameter in the response. That means the client is less able to select specific `TLSA` records, thereby vastly increasing the response size if multiple TLS services are present for the domain. The client will also have to parse each of these records.

Solution #2 advocates extra labels in the DNS request. Instead of a DNS question for `TLSA` records for `www.example.org`, the client will have to request the `TLSA` records for `_443._tcp.www.example.org`. One might recognize this scheme: the same is used for `SRV` records.

Some have also mentioned solutions using *service discovery* schemes such as `NAPTR` records[15], but it is feared that the added complexity of these records will slow down deployment of `TLSA`.

Downgrade attacks and policy Another important issue is the possibility of *downgrade attacks*. This is a matter of *policy*; should the client always connect to the secure variant of the service, and if so, how would the client know where to find this service?

Implicit policies where HTTP clients (port 80) would automatically connect to the HTTPS variant (port 443) of the service might have undesirable effects. The mailing list participants proposed a new resource record type – `HASTLSA` – which would document the secure and insecure services available at a host. Choice of the secure or insecure variant would be at the clients' discretion.

Other policies PKIX certificates are able to set a wide range of restrictions, such as the *common name* attribute which has to be the same fully qualified domain name as the server that's being accessed. Additionally, the certificate will have an expiry date.

It is still up to the working group to decide whether these restrictions should be honoured; most likely they will be, except for the common name as that can already be deduced from the name where the `TLSA` record is retrieved.

2.3 Kaminsky Alternative

Security researcher Dan Kaminsky chose another approach. Instead of arriving to a standard through the consensus process, he delivered a working *proof of concept*. He believes that the open-source model of *release early, release often* is a suitable model for defining the new standard. Of course this means that his releases will have imperfections and a lot of his design choices are arguably made by 'gut decisions'. He does try to rationalize his decisions, and because of the 'blog' system he uses for his publications, anyone can respond to his system.

His documentation on what he calls the *Domain Key Infrastructure*[9] consists of six 'blog posts' on the system itself and after that three more posts defending DNSSEC versus DNSCurve, caching implications and alternative DNS transports such as HTTP. This section will only cover his design of the DNS record[8].

2.3.1 Defense For Choosing TXT

His first design choice is already a controversial one; he chose to embed the certificate material in the already existing `TXT` record. The IETF standards body made an RFC in late 2009 specifically recommending NOT to use `TXT` records for this purpose[17]. Kaminsky is aware of these recommendations as he cited the document himself. The most important objections to using `TXT` include the fact that those records provide no semantics to prevent collisions with other uses of the record. Another important objection are the space considerations: if every application developer would choose the easy road, clients might be flooded with a wide array of records that they have no need for. This is especially true in the case of putting public key material in DNS - these messages might get quite large in comparison to other resource records' payload.

Kaminsky argues that there are several important reasons for choosing `TXT`, one of them being backward compatibility. The other arguments seem to be based on statistics and *what everyone else is doing*.

Backward compatibility is maintained because not all (home-)routers' built-in DNS server have support for new record types. This means the messages would effectively be blocked by the router. At the other side of the spectrum, publishers of DNS data will have a more easy job because DNS web panels and back ends do not have to be updated to support a new record type.

The next argument he uses is that other projects have tried to use their own specific resource record type, but moved back to `TXT` records after failed and lacking deployment. His examples include the Sender Policy Framework (SPF), DomainKeys (DKIM), GPGs PKA and PGP Key

Retrieval and lastly FreeSWAN's IPsec implementation. The formats of these records all use a key=value system as documented in an RFC published in 1993[19].

It is clear that Kaminsky is aiming for maximum backward compatibility, but one has to wonder what good it will do if everyone will already have to take a serious look at their DNS infrastructure because of the dependency on DNSSEC.

2.3.2 Specification of TXT v=key1 record

Last section mentioned 4 other protocols that use `TXT` records:

- ``v=spf1 a -all''
- ``v=DKIM1;p=MIGfMA0G cQ2QIDAQAB''
- ``v=pka1;fpr=[#1];uri=[#2]''
- ``X-IPsec-Server(10)=192.1.1.5 AQMM3s1Q==''

All of these records except the IPsec record begin with a so called *magic cookie*; an identifying string in a blob of text, used to identify the start of a parsable section. In this case the magic cookie is `v=`, followed by the protocol and optionally its version. Kaminsky chose a similar scheme.

After the protocol and version are the protocol parameters. The parameters are defined as key-value pairs separated by spaces. Kaminsky defined the following parameters:

- `ha=[HASH ALGORITHM]`
- `h=[HASH]`
- `lh=[0|1]`
- `hr=[cert|pubkey]`
- `sts=[0|1]`
- `sn=[0|1]`

2.3.3 Parameters and features

The `ha` parameter specifies the hashing algorithm that is used on the attached data. Instead of a value from a IANA registry, as in the IETF draft, Kaminsky used a textual representation of the hashing algorithm. Currently the `sha1` algorithm is the only one explicitly listed. The `h` parameter contains the hash of the certificate, confirming to the hash algorithm specified in the `ha` parameter.

A feature called Livehashing can be enabled by setting `lh=1` in the TXT record. If livehashing is enabled, and the hash of the certificate is not specified, the client should calculate the hash of the retrieved certificate. A second lookup should be done with the hash prepended as an extra label to the domain name. If the response contains any secure record, the domain name and hash are considered to be acceptably linked. Although an extra round trip is required, this improves deployability for large sites. The `hr`, Hash Range, parameter specifies what is stored in DNS. In the case of `hr=cert`, a hash of the entire certificate is published. And if `hr=pubkey`, then only the public key is hashed.

Kaminsky also supports a `sts` parameter. A value of `1` indicates Strict-Transport-Security[6], to enforce the use of TLS when accessing web sites. This solves a problem of STS during the first connection to a site. Since there is no STS to enforce the initial connection being secure, can you trust the value for STS you receive? Thus, the specification of Kaminsky can be used to securely express TLS only connections to a server. The `sn` parameter is used to indicate that the server supports Secure Negotiation.

Chapter 3

Implementation

We implemented a working proof of concept that shows the potential of an certificate less internet through a browser add-on that checks for the availability of valid certificates over DNS using DNSSEC, and where relevant uses them and/or compares them with regularly available certificates.

In the next section we will give an overview of the current available implementations, followed by our choices regarding the browser. Finally we will conclude this chapter with the implementation details of our add-on.

3.1 Current implementations

There are only a few implementations of DNSSEC validation for web browsers. They all offer a implementation that validates DNS responses for A and AAAA record types. What we are interested in is checking the validity of a certificate using DNSSEC. Currently there is only one implementation of that, Phreeload. However, this doesn't work with together with the current web browsers. In the following sections we will present four implementations and why they are not suitable for our goals.

3.1.1 DNSSEC Validator

DNSSEC Validator is an add-on for the Mozilla Firefox web browser, which allows you to check the existence and validity of DNSSEC DNS records for domain names in the address of the page currently displayed in your browser window. The result of this check is displayed using color keys and information texts in the page's address bar[11]

The DNSSEC Validator extension is currently one of the most usable extensions for DNSSEC validation. However there is an important security issue. During the validation process it relies on the AD flag, authenticated data, being set in the DNS response packet. If the AD flag is set in a response, the name server considers all RRsets in the Answer and Authority sections of the response to be authentic. But this is just a bit that is set at the name server. Since one can usually not trust the *last hop*, from the DNS resolver to the client, the AD flag inside the response can be spoofed. The result of this add-on can therefore only be trusted if one would use it together with a local validating DNS resolver, for example Unbound[14].

Another reason for not completely basing our add-on on DNSSEC Validator is related to the validation process. Validating the DNS responses is implemented in the add-on and can therefor only be used in the web browser. Whilst we think that a validating resolver should be provided by the Operating System or as a shared library so that multiple applications can use it.

3.1.2 DNSSEC Drill: Extension for Firefox

This extension performs DNSSEC lookups for the main hostname of the current page in firefox. It uses Drill to chase the signatures up to a trusted key. The user can specify trusted keys by putting them in a directory of his choice[12] The DNSSEC Drill extension created by NLnet Labs depends on ldns[13] being present on the system. ldns is a DNS library supporting recent RFCs including DNSSEC. One of the bundled tools with ldns is drill. This tool is used by the extension to chase the DNSSEC signatures up to a trusted key.

This add-on calls the drill executable on the system and checks the exit code. Based on the exit code, the add-on can show if the domain is secured by DNSSEC. Due to this implementation method only the exit code, and not the actual DNS answer, is accessible from the add-on. This means that the browser itself can continue with a spoofed DNS answer, even though a separate DNS lookup outside the browser is perfectly valid.

3.1.3 DNSSEC-Tools

The goal of the DNSSEC-Tools project is to create a set of software tools, patches, applications, wrappers, extensions, and plugins that will help ease the deployment of DNSSEC related technologies.[20]

This set of tools provides, among others, a patch for Firefox. The patch enables DNSSEC validation of DNS lookups in the Firefox application suite (the Firefox browser, Mozilla, etc). An important difference between a patch and add-on is that a patch requires a recompilation of Firefox. While an add-on can easily be added to an existing installation.

3.1.4 Phreebird Suite - Phreeload

Phreebird is a DNSSEC proxy that operates in front of an existing DNS server (such as BIND, Unbound, PowerDNS, Microsoft DNS, or QIP) and supplements its records with DNSSEC responses. Features of Phreebird include automatic key generation, real time record signing, support for arbitrary responses, zero configuration, NSEC3 "White Lies", caching and rate limiting to deter DoS attacks, and experimental support for both Coarse Time over DNS and HTTP Virtual Channels. The suite also contains a large amount of sample code, including support for federated identity over OpenSSH. Finally, Phreeload enhances existing OpenSSL applications with DNSSEC support.[9]

Phreeload, which is included in the Phreeburd Suite, adds DNSSEC verification to OpenSSL apps. It does this by hooking into LD_PRELOAD. But browsers don't use OpenSSL, Internet Explorer uses CryptoAPI and Firefox and Chrome use Network Security Services (NSS). A different implementation is therefore still required to user DNSSEC to associate certificates with domain names in a web browser.

3.2 The browser

Our implementation will be an add-on for the Firefox web browser. The architecture of Firefox allows for easy add-on development. A few implementations for the Firefox web browser already exist. Although none of them are completely usable for our purposes, they can serve as a good starting point.

We specifically choose to only support Firefox 4.0, even though this is still in beta. Version 4.0 provides a feature to call native C functions without any glue code. This allows us to easily distribute the add-on together with libunbound, which is described in section 3.3.1, for multiple platforms. This is a useful feature since it makes it possible to create an add-on that doesn't rely on any non-standard tools or libraries being installed on the system.

3.3 Add-on

In the next sections we will describe the important details of the add-on. Section 3.3.1 will give an overview of the validating DNS resolver that is integrated in the add-on. Followed by a description of the validation methods and possible result states in section 3.3.2. Finally, section 3.3.3 outlines a problem with Strict Transport Security that is caused by two non compliant drafts.

3.3.1 Local validating DNS resolver

As we mentioned before in section 3.1.1, we can only rely on the DNS AD flag, if the validating DNS resolver is running locally. In some environments one might be able to trust the (local) path to a DNS resolver, but this is usually not the case when using your ISPs resolver or while being connected to an open WiFi network. Running a local validating DNS resolver as a server might not be desired on certain systems. An alternative to this is a stub-resolver that is linked into an application. One of the components of Unbound[14], a server daemon implementation of a validating DNS resolver, is the library API libunbound.

Our add-on uses libunbound for performing validated DNS lookups. We believe that an implantation of the functionality provided by libunbound should be available on every system. For the time being the add-on will include libunbound as a fall back mechanism if it is not provided by the system.

3.3.2 Validation methods

The add-on performs two validation methods. It checks if the domain is signed by DNSSEC and whether the DNS response is valid. If the domain is signed by DNSSEC and the connection to the web server happens over a secure channel, the validity of the certificate will also be checked. Firefox provides an interface, nsIWebProgressListener, that can be implemented to listen in on the progress associated with the loading of requests. When a new request is made, the add-on will do a lookup for the domain using libunbound and check if a valid and secured DNS response is received. Without this, all policies and validation data published in DNS can't be trusted. The flowchart in Figure A.2 shows that if a domain isn't signed by DNSSEC, the add-on doesn't continue with the validation process and gives control back to the browser. It's important to realize that any further checks wouldn't add any security in the case of a unsigned domain. If an attacker would be intercepting the connection, then the DNS responses containing the validation data can be spoofed as well.

The second validation method will be called if the connection to the web server is on a secure channel. The certificate received trough the HTTPS connection will be checked for validity. First the certificate will be validated using the WebTrust Certificate Authorities that are available in the browser. If the DNS records of the domain contain data that can be used to validate the certificate, then this is also checked. The flowchart in Figure A.3 shows the possible security states after validating the certificate. A self-signed certificate would normally not get accepted by the browser. If it can be trusted based on validated data obtained from DNS, an exception for the certificate is passed to the browser. This exception will disable the self-signed certificate warning and the end-user will be able to browse a secured version of the web page.

3.3.3 HTTP Strict Transport Security

Firefox 4 supports the IETF draft *HTTP Strict Transport Security (HSTS)*[6]. HSTS is a mechanism enabling Web sites to declare themselves accessible only via secure connections. Besides the Strict-Transport-Security (STS) header set by a web server during an HTTPS response, Firefox also provides an interface for add-ons to enable STS for sites that don't set the STS header themselves.

A problem with the current HSTS draft is that it is not compliant with the in Section 2.2.2 described draft *Using Secure DNS to Associate Certificates with Domain Names For TLS* or the specification described in Section 2.3.2. The HSTS policy states that the user agent should terminate any secure transport connection attempts upon any and all secure transport errors or warnings, including those caused by a site presenting self-signed certificates. This conflicts with the other drafts that state that a self-signed certificate should be accepted if a valid corresponding fingerprint record is found under the domain name of the web server.

The HSTS implementation in Firefox 4 complies to the previous statement and doesn't allow exceptions for self-signed certificates when STS is enabled for a domain. This has some consequences for our add-on. If a record under a domain name specifies that STS should be enabled, it can't be followed without some further checks. The domain can only be marked as STS when the browser already accepts the certificate, signed by a Certificate Authority. If the certificate presented by the web server can only be validated using DNSSEC, the add-on adds only adds an exception to accept the certificate. The domain will be redirected to HTTPS, but won't be marked as STS enabled, since that conflicts with accepting the self-signed certificate based on validated DNS records.

Not being able to enable STS when specified in a DNS record also affects resources loaded by the main document, such as images or style sheets. If STS could be enabled, all these sub documents are also forced to use a secure connection.

Chapter 4

Conclusion

At the time of writing, the IETF is defining the standard described in section 2.2. Many options are being considered while working towards a complete design. Although a consensus on all these choices takes a long time, it will in the end lead to a well considered standard. In contrast, Dan Kaminsky showed that using a *release early, release often* approach has advantages as well. His choice of a `TXT` record could be considered bad, but did work out to a quick and very practical solution. Taking the quick route at this moment has its advantages. Since the subject is very active at this moment, having a quick practical solution showed what is possible and which choices are important to consider in the process.

With DNSSEC being a requirement for associating certificates with domain names using the Domain Name System, an important problem had to be solved. Namely the *last hop* of a secure DNS system, from the end client to the DNS resolver. Relying on the AD flag is not sufficient, especially when taking open WiFi networks in consideration. Running a full local DNS resolver is not desirable on every system, neither is having every application implementing its own DNSSEC validation methods. We can conclude that DNSSEC support should be provided by the operating system or as a library API that can be shared by multiple applications.

From the development of our proof of concept add-on we can conclude that it's not that difficult to integrate a local validating resolver, in our case libunbound, into an add-on or application. Once the standard on using secure DNS to associate certificates with domain names is nearing completion, the implementation can be adopted quickly by web browsers and other applications.

4.1 Future work

Future research needs to be done on the transitioning path from checking certificates using signing authorities to using the secure channel provided by the DNSSEC infrastructure. If there's no valid certificate association published in DNS, should a client application always fall back to the current method using Certificate Authorities? And what are the implications of having this fall back mechanism, do we want to end up with a weakest link situation?

There's no future in domain validated certificates; a more secure alternative will exist soon. However, associating certificates with domain names using secure DNS only confirms the domain name. Certificate Authorities can add value by confirming the actual entity behind a domain. We think that the hypothetical `.trusted` could add value against phishing attacks. Inside this top-level domain CA's could publish Extended Validation certificates in a secure way. Future research could explore the organizational structure for this TLD and different certificate 'class' sub domains(ie. `*.bank.trusted` to indicate domains safe for online banking).

Appendix A

Flowcharts

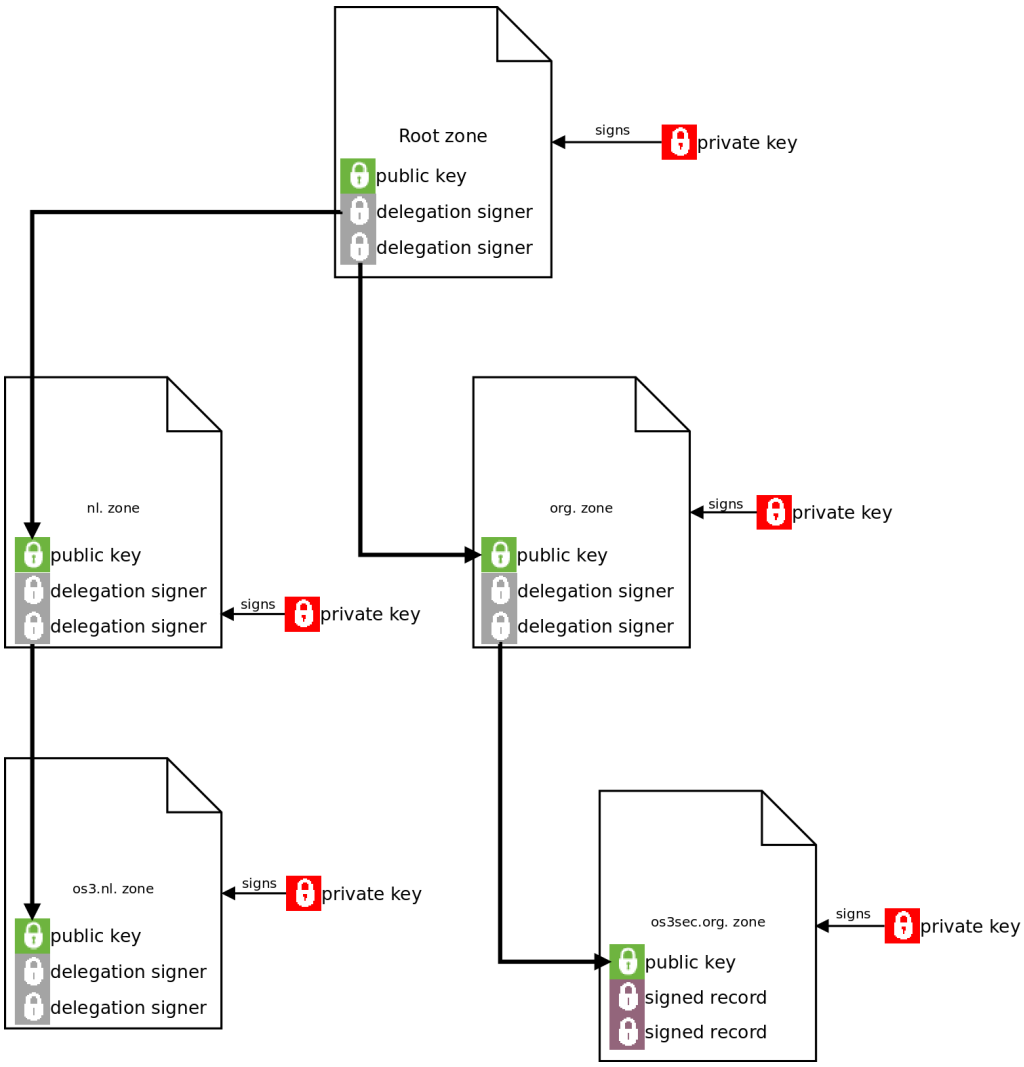


Figure A.1: Simplified DNSSEC chain of trust

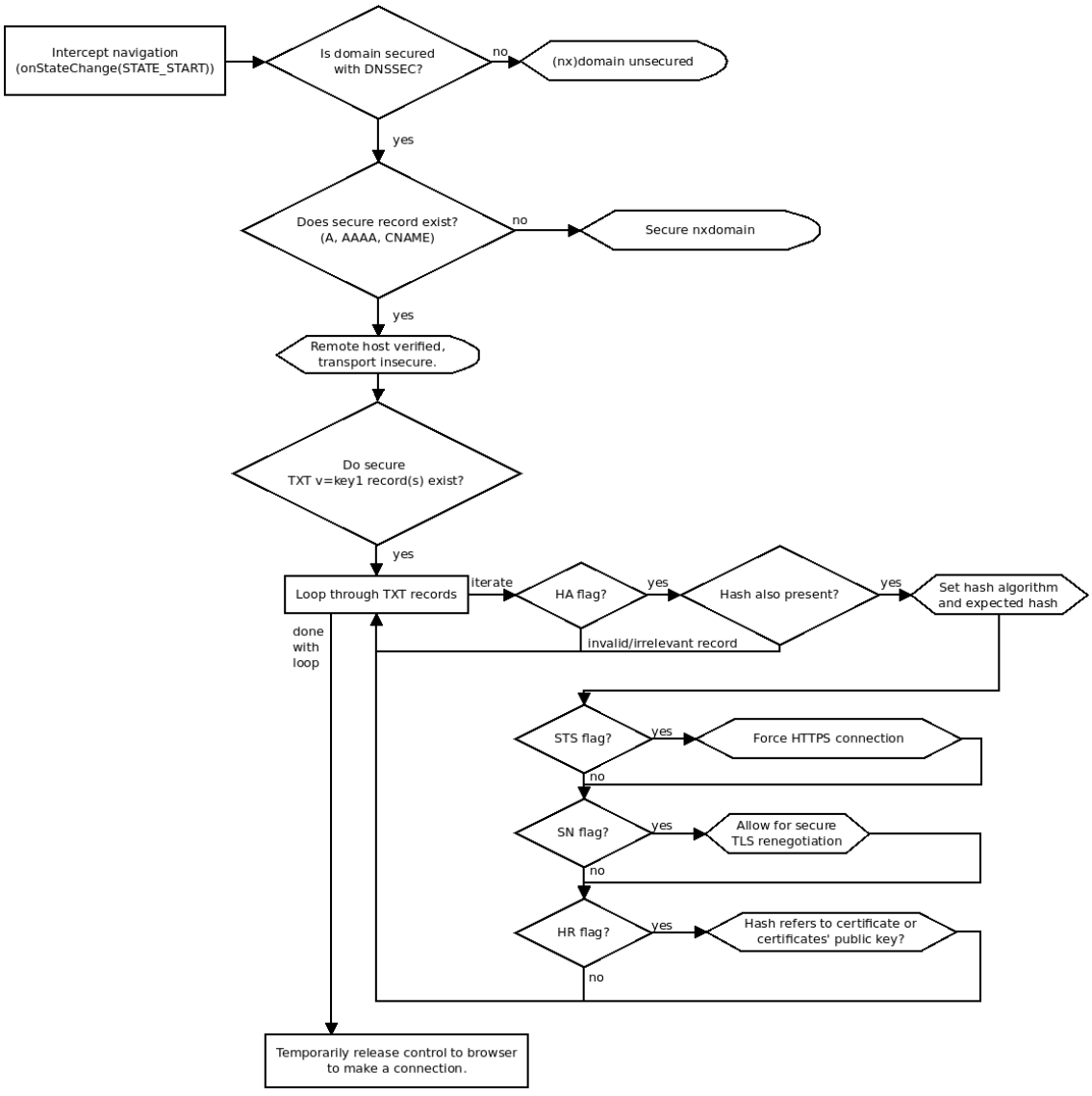


Figure A.2: DNS flowchart

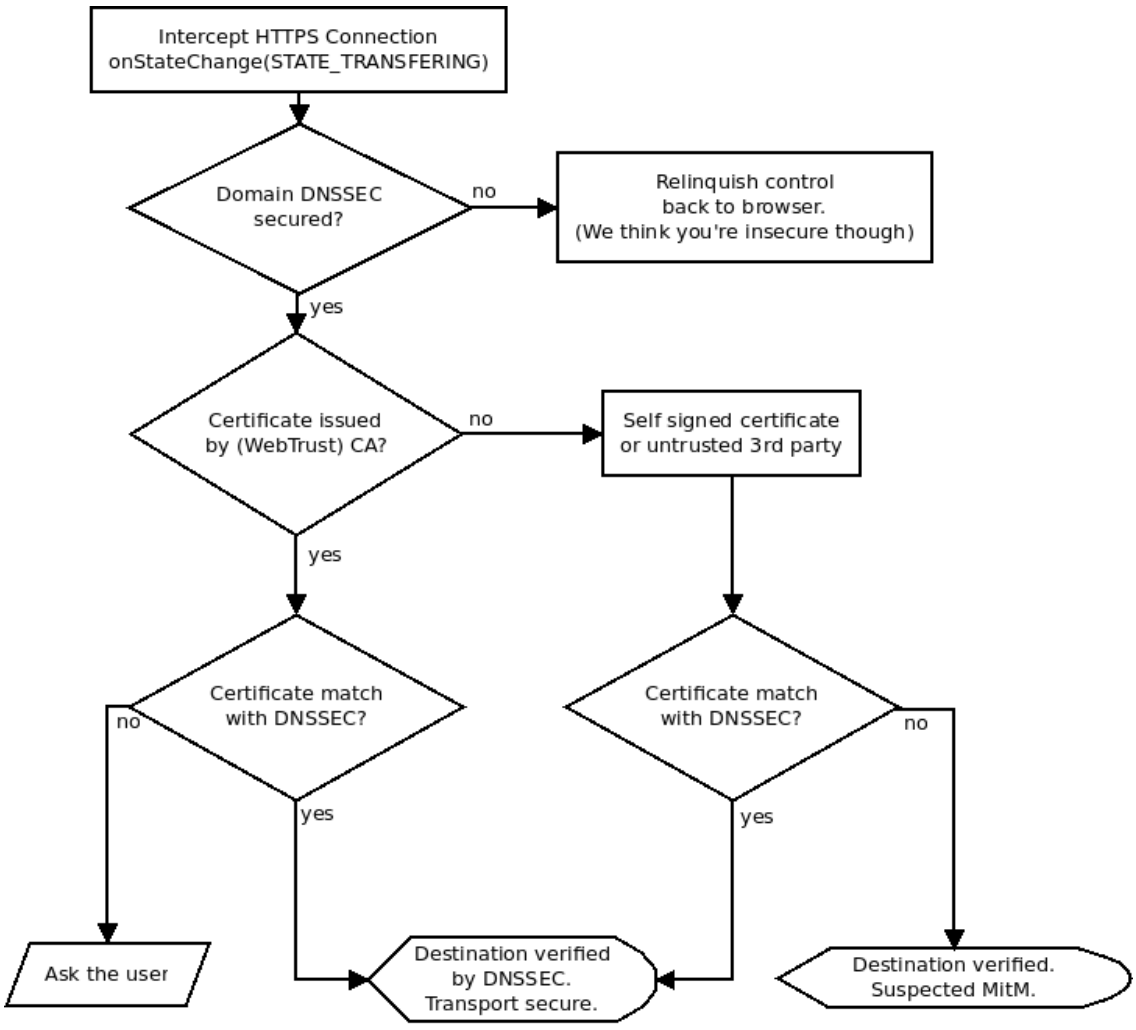


Figure A.3: State flowchart

Appendix B

Example DNSSEC signature chase

```
plange@fx160-24:~$ dig +sigchase www.os3sec.org
;; RRset to chase:
www.os3sec.org.    600 IN  CNAME web.os3sec.org.


;; RRSIG of the RRset to chase:
www.os3sec.org.    600 IN  RRSIG CNAME 8 3 600 20110306042501 20110204042501 771 os3sec.org. Sf535bnI3Mj8Vz



Launch a query to find a RRset of type DNSKEY for zone: os3sec.org.

;; DNSKEYset that signs the RRset to chase:
os3sec.org.    600 IN  DNSKEY  256 3 8 AwEAAdfNZBq/QmeXgR+4NSFvDIczaha1qTs3udqsMwuW/wGedZIEIY51 VfamB8ErYyo
os3sec.org.    600 IN  DNSKEY  257 3 8 AwEAAcfrYpMPBosrQdMLAC6wxHzEwli2YvfDlN1VmKNbGIp1Oz2pGNKC 6adBMxAbdLv


;; RRSIG of the DNSKEYset that signs the RRset to chase:
os3sec.org.    600 IN  RRSIG DNSKEY 8 2 600 20110306042501 20110204042501 771 os3sec.org. ywYW3fciAYpGXHHD
os3sec.org.    600 IN  RRSIG DNSKEY 8 2 600 20110306042501 20110204042501 6694 os3sec.org. l4NY7rRUfJTyoR67



Launch a query to find a RRset of type DS for zone: os3sec.org.

;; DSset of the DNSKEYset
os3sec.org.    76493 IN  DS  6694 8 2 BB2E6ECC8F31007B33B09121F501ED6A5F4804AD0C559FB230542EF9 A27A1F3A


;; RRSIG of the DSset of the DNSKEYset
os3sec.org.    76493 IN  RRSIG DS 7 2 86400 20110215155636 20110201145636 34260 org. R4iL6MI5ybP8uf0JyNVlh0




;; WE HAVE MATERIAL, WE NOW DO VALIDATION
;; VERIFYING CNAME RRset for www.os3sec.org. with DNSKEY:771: success
;; OK We found DNSKEY (or more) to validate the RRset
;; Now, we are going to validate this DNSKEY by the DS
;; OK a DS valids a DNSKEY in the RRset
;; Now verify that this DNSKEY validates the DNSKEY RRset
;; VERIFYING DNSKEY RRset for os3sec.org. with DNSKEY:6694: success
;; OK this DNSKEY (validated by the DS) validates the RRset of the DNSKEYs, thus the DNSKEY validates the
;; Now, we want to validate the DS :  recursive call


Launch a query to find a RRset of type DNSKEY for zone: org.

;; DNSKEYset that signs the RRset to chase:
org.       900 IN  DNSKEY  256 3 7 AwEAAZTErUFsgGZliJ7xFCdRUIIrvuz5LU8wgryBBNZXZ/xkhZ/4hw/D L8dBvBVNNXeKB
org.       900 IN  DNSKEY  257 3 7 AwEAAYpYfj3aaRzzkxWQqMdl7YExY81NdYSv+qayuZDodnZ9IMh0bwMc YaVUdzNAbVeJ8g
org.       900 IN  DNSKEY  257 3 7 AwEAAZTjbIO5kIpxWUtyXc8avsKyHIIZ+LjC2Dv8naO+Tz6X2fqzDC1b dq7HlZwtkaqTkMV
org.       900 IN  DNSKEY  256 3 7 AwEAAZMKvhAE5BARHVleVsDcGRBQBFYdfAbhixOI9a3tZ4av7wX0HB6/ ZUWDp5m+WeUoR/


;; RRSIG of the DNSKEYset that signs the RRset to chase:
```

```
57 org.       900 IN  RRSIG DNSKEY 7 1 900 20110215155636 20110201145636 21366 org. PjQQCVPqiLJBF95iGjP3wntwRo
58 org.       900 IN  RRSIG DNSKEY 7 1 900 20110215155636 20110201145636 34260 org. FCc6z/OzEZDBXML5lL//VbKw0
59
60
61
62 Launch a query to find a RRset of type DS for zone: org.
63
64 ;; DSset of the DNSKEYset
65 org.       35434 IN  DS  21366 7 1 E6C1716CFB6BDC84E84CE1AB5510DAC69173B5B2
66 org.       35434 IN  DS  21366 7 2 96EEB2FFD9B00CD4694E78278B5EFDAB0A80446567B69F634DA078F0 D90F01BA
67
68
69 ;; RRSIG of the DSset of the DNSKEYset
70 org.       35434 IN  RRSIG DS 8 1 86400 20110210000000 20110202230000 21639 . unpuP88/6v/CvH0WFlZsV4nKPkU
71
72
73
74
75 ;; WE HAVE MATERIAL, WE NOW DO VALIDATION
76 ;; VERIFYING DS RRset for os3sec.org. with DNSKEY:34260: success
77 ;; OK We found DNSKEY (or more) to validate the RRset
78 ;; Now, we are going to validate this DNSKEY by the DS
79 ;; OK a DS valids a DNSKEY in the RRset
80 ;; Now verify that this DNSKEY validates the DNSKEY RRset
81 ;; VERIFYING DNSKEY RRset for org. with DNSKEY:21366: success
82 ;; OK this DNSKEY (validated by the DS) validates the RRset of the DNSKEYs, thus the DNSKEY validates the
83 ;; Now, we want to validate the DS :  recursive call
84
85
86 Launch a query to find a RRset of type DNSKEY for zone: .
87
88 ;; DNSKEYset that signs the RRset to chase:
89 .      169131  IN  DNSKEY  256 3 8 AwEAAb5gVAzK59YHDxf/DnswfO1RmbRZ6W16JfhFecfI+EUHRXPWlXDi 47t2FHaKyMMER
90 .      169131  IN  DNSKEY  257 3 8 AwEAAagAIKlVZrpC6Ia7gEzahOR+9W29euxhJhVVLOyQbSEW0O8gcCjF FVQUTf6v58fLjwB
91
92
93 ;; RRSIG of the DNSKEYset that signs the RRset to chase:
94 .      169131  IN  RRSIG DNSKEY 8 0 172800 20110214235959 20110131000000 19036 . US/x2AqVVg24lZSXALdxBdZFa8
95
96
97
98 Launch a query to find a RRset of type DS for zone: .
99 ;; NO ANSWERS: no more
100
101 ;; WARNING There is no DS for the zone: .
102
103
104
105 ;; WE HAVE MATERIAL, WE NOW DO VALIDATION
106 ;; VERIFYING DS RRset for org. with DNSKEY:21639: success
107 ;; OK We found DNSKEY (or more) to validate the RRset
108 ;; Ok, find a Trusted Key in the DNSKEY RRset: 19036
109 ;; VERIFYING DNSKEY RRset for . with DNSKEY:19036: success
110
111 ;; Ok this DNSKEY is a Trusted Key, DNSSEC validation is ok: SUCCESS
```

Bibliography

[1] R. Austein D. Atkins. Threat analysis of the domain name system (dns). Technical report, IETF Request For Comments, 2004. https://datatracker.ietf.org/doc/rfc3833/.

[2] S. Farrell D. Cooper, S. Santesson. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. Technical report, IETF Request For Comments, 2008. https://datatracker.ietf.org/doc/rfc5280/.

[3] Electronic Frontier Foundation. The eff ssl observatory. http://www.eff.org/observatory, 2010.

[4] WebTrust Certification Authorities Advisory Group. Guidelines for the issuance and management of extended validation certificates, 2008. version 1.1.

[5] IETF. dane charter. http://datatracker.ietf.org/wg/dane/charter/, 2010.

[6] A. Barth J. Hodges, C. Jackson. Http strict transport security (hsts). Technical report, Internet Engineering Task Force, 2011. http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-00.

[7] S. Josefsson. Storing certificates in the domain name system (dns). Technical report, IETF Request For Comments, 2006. https://datatracker.ietf.org/doc/rfc4398/.

[8] Dan Kaminsky. The dnssec diaries. http://dankaminsky.com/2010/12/13/dnssec-ch1/, December 2010.

[9] Dan Kaminsky. Introducing the domain key infrastructure. In *Phreebird suite*, 2010.

[10] Amit Klein. Bind9 dns cache poisoning. Technical report, 2007. http://www.trusteer.com/bind9dns.

[11] CZ.NIC Labs. Dnssec validator. http://www.dnssec-validator.cz/, 2010. version 1.0.4.

[12] NLnet Labs. Dnssec drill: Extension for firefox. http://nlnetlabs.nl/projects/drill/drill_extension.html, 2010. version 0.7.1.

[13] NLnet Labs. ldns. http://www.nlnetlabs.nl/projects/ldns/, 2010.

[14] NLnet Labs. Unbound. http://www.unbound.net/, 2011. version 1.4.8.

[15] R. Daniel M. Mealling. The naming authority pointer (naptr) dns resource record. Technical report, IETF Request For Comments, 2000. https://datatracker.ietf.org/doc/rfc2915/.

[16] P. Mockapetris. Domain names - implementation and specification. Technical report, IETF Request For Comments, 1987. https://datatracker.ietf.org/doc/rfc1035/.

[17] P. Koch P. Faltstrom, R. Austein. Design choices when expanding the dns. Technical report, IETF Request For Comments, 2009. https://datatracker.ietf.org/doc/rfc5507/.

[18] J. Schlyter P. Hoffman. Using secure dns to associate certificates with domain names for tls. Technical report, Internet Engineering Task Force, 2011. http://tools.ietf.org/html/draft-ietf-dane-protocol-03.

[19] R. Rosenbaum. Using the domain name system to store arbitrary string attributes. Technical report, IETF Request For Comments, 1993. https://datatracker.ietf.org/doc/rfc1464/.

[20] DNSSEC Tools. Dnssec-tools. https://www.dnssec-tools.org/resources/tools.html, 2010. version 1.8.

[21] David Ulevitch. Opendns. http://www.opendns.com/.