Enabling DNSSEC in Open Source Applications

Wes Hardaker, Suresh Krishnaswamy
{hardaker,suresh}@sparta.com
SPARTA, Inc.

Abstract—**The Domain Name System (DNS) [1] [2] has been recently improved by the addition of DNS security extensions (DNSSEC) [3] [4] [5]. These improvements secure DNS against information forgery, modification and other attacks [6]. The DNS infrastructure needs to be upgraded to take advantage of the benefits offered by DNSSEC. Servers will need to serve DNSSEC enabled records and applications will need to look for and process these new security records. This paper discusses the advantages of supporting DNSSEC directly within end-system applications and the intricacies involved in retrofitting existing applications with DNSSEC support. The experiences and benefits achieved when upgrading two open-source packages is described.**

I. Introduction

The Domain Name System (DNS) [1] [2] is a critical Internet infrastructure protocol that translates human-friendly domain names (like "www.example.com") to numeric Internet Protocol (IP) addresses (like "192.0.2.1"). The DNS has been part of the Internet infrastructure for over 20 years. However, it was not originally designed with security in mind and was left vulnerable to message modification and data forgery attacks. The DNS Security Extensions (DNSSEC) [3] [4] [5] fix the security flaws in the original DNS protocol.

The DNS Security Extensions provides origin authentication, data integrity, and authenticated denial of existence (the ability to reliably detect through a proof of non-existence, if a name or type in the DNS data is intentionally missing). Consumers of DNS information can check cryptographic signatures present on data to gain assurance that the data was created by an authorized entity and that it was not modified in any way from the time that the authorized entity created it. DNSSEC is just beginning to achieve wide-spread deployment. In order for deployment to be considered complete, all DNS data producers need to add DNSSEC security records to their zone data and all DNS data consumers must request DNSSEC responses and must be able to validate those responses in order to ensure that the returned records have not been tampered with or forged.

There are multiple ways that DNSSEC deployment can be achieved near the DNS data consuming clients. Validation of responses can either be done through a trusted security-aware recursive name server [3] near the client or by client applications themselves. This paper focuses on the details and benefits of implementing DNSSEC validation directly within DNS client applications.

This paper is organized as follows: Section II describes the motivation behind making DNS applications perform validation themselves. Section III discusses the existing DNS API and the changes required to it in order to support DNSSEC. It focuses on the new DNSSEC-enabled API provided by the DNSSEC-Tools software package [7]. Section IV briefly discusses modern applications' use of DNS within their architectures and discusses the types of modifications that are needed in applications to support the new DNSSEC-specific API. It also provides details on instrumenting two example applications: Firefox [8] and OpenSSH [9] with DNSSEC capability. Section V discusses lessons learned from this work, section VI discusses future work and section VII discusses conclusions reached.

23 January 2011

II. Motivation for DNSSEC Instrumented Applications

There are two ways of doing DNSSEC validation near the DNS data-consuming end. One method is to deploy a security-aware recursive name server within the network closest to the end-systems that are requesting answers to DNS queries. This recursive name server would then perform all the work on behalf of the client and check the result before returning a response to the client. The other possibility is to have the end-systems themselves request and process all DNS security information. Each method has its advantages and disadvantages.

The benefits to deploying DNSSEC validation on a recursive name server center on the fact that it simplifies the configuration and maintenance of DNSSEC-related information. When trust anchors need to be added, removed or modified they can be changed entirely within the few hosts that offer recursive name service support to the clients. End-system libraries, applications and configuration require no changes to take advantage of a security-aware recursive name server that is protecting it from DNS information modification. In contrast, all validating end-devices must be configured with a list of trust anchors and policy definitions if DNSSEC is performed within end-systems.

A downside of centralization, though, is that the network between the validating recursive name server and the end-system needs to be trusted. Although DNS data is protected between the distributing source and the validating name server, there is no protection provided between the validating resolver and the end-system. There are multiple ways to protect the network infrastructure using IPsec, DNS TSIG [10], physical security and other protection mechanisms but these add back in end-system configuration complexity, which defeats one of the primary objectives of centralized validation.

Another benefit to performing validation on a recursive name server is that the caching aspect of the resolver reduces network load since each client does not need to request the larger DNSSEC related resource records. However, this is not a significant problem even when validation is performed on end-systems. Validating clients can still use caching resolvers when requesting DNSSEC data so the increase in traffic required by validating end-systems can be completely localized to the infrastructure between the end-systems and the caching resolver where bandwidth is typically plentiful.

An issue with performing validation on a caching resolver is that the only errors that the normal DNS protocol can return to the client are those associated with a successful validation condition or a generic error. This doesn't enable the client to determine the reason for the look-up failure (e.g. a regular DNS look-up failure v.s. a DNSSEC specific error). Only direct validation on the end-system application can take advantage of additional details associated with the validation process. As will be discussed later in this paper (Section IV), these additional error indications can be very helpful in communicating problem events to the end user through the use of better graphical status icons, color codings or error messages.

Finally, performing validation on a recursive name server provides no local control on the end-system, which may have differing policy requirements. If an end-system had a greater trust requirement for some component of the Internet it would have no way of ensuring the validating recursive name server was appropriately checking the results from that Internet segment. When validation is done within end-systems, each application can have its own required security policies that could differ from other applications on the same system. Although this scenario is less frequently needed, it is still important to support.

While there are trade-offs associated with performing in-application validation, the authors believe that the benefits associated with application-level validation, particularly with respect to enabling fine-grained local control and providing better error messages to the end-user, far outweigh any of its administrative burdens.

III. DNSSEC API Description

DNSSEC presents a number of new status codes for application consumption. Certain status codes indicate the aggregated validity of the complete DNS message returned in response to a DNS query. Since a DNS response can be comprised of multiple resource record sets, certain validation status codes also indicate the validity of individual resource record sets in the response. Finally, the validation status code for each resource record set is determined through the validation sub-states for the individual elements in the DNSSEC authentication chain for that resource record set.

While some applications may wish to use the full set of individual DNSSEC validator status values to communicate detailed results of validation processing to the end-user, one can be reasonably certain that not all applications will want to use more than the single aggregated status value. For example, an application that uses existing DNS resolver functions may benefit from having a smaller number of DNSSEC validation states to deal with, since this is arguably easier to retrofit into its existing code base.

DNS lookups are finely ingrained into Internet applications available today. Some of these applications have been rarely modified over the years. In order for DNSSEC to be easily retrofitted into such applications it is important that any change introduced by DNSSEC to the resolver API be well-defined and have a small code footprint, in order to minimize the likelihood of introducing new vulnerabilities through programming errors related to implementing these changes. In this light, it is useful to examine some of the widely-used DNS resolver functions and consider the possible ways in which they may be modified in order to support DNSSEC.

The subsequent sub-sections define a DNSSEC-enabled API set that is suitable as a replacement API for the traditional DNS API. The API is designed to be generic such that multiple validation libraries could use the API to expose DNSSEC support to their users. The validation library in the DNSSEC-Tools software package [7] implements this API and acts as both a solution for application developers and as a reference release for validation library developers.

A. DNSSEC Enabling Historical DNS Functions

The getaddrinfo() function [11] is a replacement function for the obsoleted name to address function, gethostbyname(). Its prototype is provided below.

```
int getaddrinfo(const char *hostname,
        const char *servname,
        const struct addrinfo *hints,
        struct addrinfo **res);
```

The getaddrinfo() function returns a linked list of addrinfo structures for a given host name and service name, where each element in the linked list corresponds to a returned socket address. It returns 0 on success and an error code on failure. An application that calls getaddrinfo() checks the return value for error conditions and either, as a rare case, propagates these errors up or, more typically, returns a generic error to some higher level module for further processing. An application must invoke the freeaddrinfo() function to release the memory allocated for the addrinfo linked list after use.

There are a couple of choices for retrofitting DNSSEC capabilities into getaddrinfo(). The first alternative is that of adding a new validation status element to the existing addrinfo structure. To see why this is beneficial, one would need to consider how the addrinfo elements are constructed from the DNS responses. The addrinfo linked list contains a number of socket address structures corresponding to the addresses that match the query name in the DNS. Depending on the options passed to getaddrinfo, these may be a combination of IPv4 and IPv6 addresses which are represented by distinct resource record types in the DNS. Any canonical names in the addrinfo linked list are also returned in separate resource record sets. As noted earlier, each resource record set can be associated with its own validation status; thus, a new validation status element added to the addrinfo structure enables an application to determine the validity of individual resource record sets in the returned response and therefore selectively use only those components that validate successfully.

The disadvantage with adding a new member to the addrinfo structure is that it increases the number of places in the application code that needs to be modified in order to add DNSSEC support. Many applications eventually end up passing the addrinfo linked list returned by getaddrinfo() to a number of additional functions before they finally call the freeaddrinfo() function. In order to add DNSSEC support to such applications, every function that uses the addrinfo structure would need to recognize the modified addrinfo structure and handle the error condition arising from encountering elements in the list that do not validate successfully. The freeaddrinfo() function and all the parts of the code that use this function would also need to be modified in order to handle the modified addrinfo structure.

An alternative to modifying the addrinfo structure is that of having an additional status code returned as an argument to the getaddrinfo() function. In this approach the application loses the ability to check the DNSSEC validity status of individual answers in the addrinfo linked list; however, application changes are kept to a minimum. From the perspective of having a replacement function that can be easily integrated into existing applications, the second approach is more desirable. It can also be argued that applications that need to obtain detailed validation status information for each resource record set can always use one of the new DNSSEC-capable functions discussed in Section III-B. Thus, for a DNSSEC-capable version of getaddrinfo() the DNSSEC validator library in the DNSSEC-Tools software package adopts the second approach.

The prototype for such a modified getaddrinfo() function is provided below.

```
int val_getaddrinfo(val_context_t *ctx,
        const char *hostname,
        const char *servname,
        const struct addrinfo *hints,
        struct addrinfo **res,
        val_status_t *val_status);
```

In the spirit of keeping the application code change to a minimum, the set of possible return values from val_getaddrinfo() is kept identical to that of getaddrinfo(). The DNSSEC validation status is returned in the val_status argument. This value represents the consolidated DNSSEC validation status for all answers returned by the val_getaddrinfo() function. Applications can use this status code to decide if the set of answers corresponds to a validated answer, a locally trusted answer (through local policy rules), an invalid answer, or either a provable or locally trusted non-existent DNS name or type code.

Local policy plays an important role in deciding the validation outcome for a DNS query. The ctx element in the val_getaddrinfo() function is a handle to the validator context which references the particular validator policy in use. This value may simply be set to NULL for those applications that wish to just use the default system-wide DNSSEC validation policy. Section III-C describes the DNSSEC validator policy considerations in greater detail.

The above template for a DNSSEC-capable getaddrinfo() replacement can be easily extended to other historical DNS API functions. Prototypes for getnameinfo() and res_query() replacements are provided below.

```
int val_getnameinfo(val_context_t *ctx,
        const struct sockaddr *sa,
        socklen_t salen,
        char *host,
        size_t hostlen,
        char *serv,
        size_t servlen,
        int flags,
        val_status_t *val_status);

int val_res_query(val_context_t *ctx,
        const char *domain_name,
        int class,
        int type,
        u_char *answer,
        int anslen,
        val_status_t *val_status);
```

While the getaddrinfo() and getnameinfo() functions supercede older gethostbyname() and gethostbyaddr() functions, it should be noted that a large number of applications still use these obsoleted functions. DNSSEC-capable versions of gethostbyname() and gethostbyaddr() can also be defined in a similar manner.

B. New DNSSEC-specific APIs

It is expected that the average application will more likely wish to use the simple resolving APIs previously discussed in combination with additional DNSSEC-specific validation result checking APIs. This combination provides a minimal upgrade path from being a DNSSEC-agnostic application to a DNSSEC-aware application. A more sophisticated and powerful DNSSEC-enabled resolving API is also provided by the DNSSEC-Tools library. These additional fine-grained functions provide applications the ability to access the entire chain of data and the validity results of each DNS request/response pair. Some applications may wish to make use of these more detailed API routines in order to provide sufficient debugging and troubleshooting utilities for end-users or administrators.

1) Validation Result Checking APIs: The DNSSEC-Tools API offers a number of convenience functions for testing the validity of returned responses, regardless of whether they are aggregated validation status of the functions described in

III-A or the individual results returned from the fine-grained resolving APIs. The prototypes for these validation-checking functions are provided below. A return value greater than 0 indicates a true condition; other values indicate a false condition.

```
int val_isvalidated(val_status_t status);

int val_istrusted(val_status_t status);

int val_does_not_exist(
        val_status_t status);
```

An application can use the val_isvalidated() function to test if the returned validation status code represents a value that is indicative of a completely validated DNSSEC authentication chain. Since a DNSSEC validator can be configured to trust certain answers even when they have not been confirmed to have a completely valid DNSSEC authentication chain, an application can also use the val_istrusted() function to determine if a validation status satisfies this lower bound of trust. The val_does_not_exist() function is used to determine if the status code corresponds to one of the non-existence states.

2) Fine-Grained DNSSEC-enabled resolving APIs: The val_get_rrset() function allows an application to inspect the DNSSEC validation status of each resource record set returned in response to a query for a given name, class and type. The results are returned in a linked list of val_answer_chain structures. Each element contains a pointer to a rr_rec structure, which encapsulates a sequence of resource record length and value tuples. Since the different resource records returned in response to a query for a given name, class and type may also have different names (if name aliases were followed) or types (if the query type code was "any"), these fields are duplicated in the val_answer_chain structure to reflect the exact values returned in the resource record set. The flags argument to the val_get_rrset() function is provided for future use and is normally set to 0. The val_free_answer_chain() function can be used by the application to release the memory allocated to the val_answer_chain linked list after use. The function prototypes and relevant structure definitions are provided below.

```
int val_get_rrset(val_context_t *ctx,
        const char *name,
        int class,
        int type,
        u_int32_t flags,
        struct val_answer_chain **ans);

void val_free_answer_chain(
        struct val_answer_chain *ans);

struct val_answer_chain {
    val_status_t val_ans_status;
    char *val_ans_name;
    int val_ans_class;
    int val_ans_type;
    struct rr_rec *val_ans;
    struct val_answer_chain *val_ans_next;
};

struct rr_rec {
    size_t rr_length;
    u_char *rr_data;
    struct rr_rec *rr_next;
};
```

The val_resolve_and_check() function enables applications to view the DNSSEC status information at an additional level of detail. It provides DNSSEC authentication chain details for each resource record set (or proof of non-existence) returned in response to a query for a given name, class and type. In addition, information for each record set (such as the time to live counter, the section from where the response was received, the server that returned it, and the individual resource records and their digital signatures) are also returned. The function prototypes and relevant structure definitions are provided below.

```
int val_resolve_and_check(
        val_context_t *ctx,
        char *domain_name,
        int class,
        int type,
        u_int32_t flags,
        struct val_result_chain **res);

void val_free_result_chain(
        struct val_result_chain *res);

struct val_result_chain {
    val_status_t val_rc_status;
    char *val_rc_alias;
    struct val_rrset_rec *val_rc_rrset;
    struct val_authentication_chain *
        val_rc_answer;
    int val_rc_proof_count;
    struct val_authentication_chain *
        val_rc_proofs[MAX_PROOFS];
    struct val_result_chain *val_rc_next;
};

struct val_authentication_chain {
    val_astatus_t val_ac_status;
    struct val_rrset_rec *val_ac_rrset;
    struct val_authentication_chain *
        val_ac_trust;
};

struct val_rrset_rec {
    int val_rrset_rcode;
    char *val_rrset_name;
    int val_rrset_class;
    int val_rrset_type;
    long val_rrset_ttl;
    int val_rrset_section;
    struct sockaddr *val_rrset_server;
    struct val_rr_rec *val_rrset_data;
    struct val_rr_rec *val_rrset_sig;
};

struct val_rr_rec {
    size_t rr_rdata_length;
    u_char *rr_rdata;
    struct val_rr_rec *rr_next;
    val_astatus_t rr_status;
};
```

The val_resolve_and_check() function returns its results in a linked list of val_result_chain structures. The val_authentication_chain structure contains the DNSSEC authentication chain details, the val_rrset_rec structure provides the detailed resource record set information, and the val_rr_rec structure points to a sequence of tuples containing resource record and signature data. The val_free_result_chain() function can be used by the application to release the memory allocated to the val_result_chain linked list after use.

Validation status values are available for each resource record set, for each element in the authentication chain for a given resource record set, for each signature that is processed to determine validity of data in a resource record set, and for each key or key hash data that is used for constructing a DNSSEC authentication chain. Applications can use the p_val_status() and p_ac_status() functions to convert these validation status values to equivalent string representations for user consumption.

```
const char *p_val_status(
        val_status_t status);

const char *p_ac_status(
        val_astatus_t status);
```

The val_get_rrset() and val_resolve_and_check() functions return 0 on success and an error code on failure. Applications can use the p_val_err() function to convert these error codes to equivalent string representations for user consumption.

```
const char *p_val_err(int err);
```

It is useful to note that a validator library that defines these functions may implement the val_get_rrset() function by invoking the val_resolve_and_check() function internally. If the VAL_QUERY_NO_AC_DETAIL flag is set in val_resolve_and_check(), authentication details for answers and proofs of non-existence are not returned in the val_result_chain structure. val_get_rrset() and val_resolve_and_check() are, then, functionally equivalent with the only exception being that val_get_rrset() will not return auxillary resource record sets associated with the response (for e.g. intermediate records in an alias chain).

C. DNSSEC Validator Policy

In the DNSSEC-Tools package, DNSSEC validator policy is stored in local system configuration (e.g. */etc/dnsval.conf*). As noted earlier, the particular DNSSEC validator policy in use by a validator impacts the validation results that it returns. Different environments may require different policy settings. The most basic of validator policy settings is the list of DNSSEC trust anchors, which typically, is simply the public key for the DNS Root zone but could also include a list of enterprise-specific internal trust anchors. Since absolute time plays an important role in DNSSEC whereas it did not for plain DNS, another useful policy setting to work around transient errors in system time or signature validity periods is the acceptable clock-skew that a validator could use while evaluating inception and expiration times on DNSSEC signatures.

The validation library in the DNSSEC-Tools software package also defines a policy setting for explicitly stating the security expectation for particular domains. This setting is useful in cases where a user wishes to selectively ignore DNSSEC processing for certain domains when it is known that no DNSSEC secure entry points for those domains exist, or to mark particular domains as untrusted. An application is then able to differentiate between a locally trusted (but not validated) status code and other resolution or validation failure codes, and can thus make a better decision on how to use a particular answer.

The term ”provably-insecure” [3] in DNSSEC is an unfortunate misnomer. While it does refer to the condition where DNSSEC is not enabled for a particular zone, it does not imply by any means that the zone data is insecure or unfit for use by an application. A provably-insecure state is usually reached through an assertion made by the parent zone that can be independently validated by DNSSEC. However, this condition is functionally equal to the case where a zone operator has set the DNSSEC security expectation for a zone to ’ignore’ and has thus instructed the validator not to verify the data. The validation library in the DNSSEC-Tools software package defines a policy setting that allows a validator operator to make a choice on whether the provably-insecure state should be considered as trusted or not. It should be noted that the val_isvalidated() function will still return a false condition for provably insecure results that are configured to be trusted through local policy settings.

While most applications on a system would use a consistent system-wide DNSSEC validator policy, it can also be envisioned that certain applications on that system may want to use their own customized DNSSEC validator policy settings for evaluating DNSSEC results. The validator context is the application’s handle to the validator policy. An application can create a new validator context using the val_create_context() function, using the scope argument to guide the validator in deciding the particular policy settings to associate with that handle. The manner in which the scope argument is used within the system configuration to identify specific validator policy settings is implementation-specific. Applications can use the val_free_context() function to release the memory allocated by the val_create_context(). The prototypes for these functions are provided below.

```
int val_create_context(const char *scope,
        val_context_t **ctx );

void val_free_context(val_context_t *ctx);
```

The validation library in the DNSSEC-Tools software package defines a hierarchical representation of policy labels for the scope parameter and a mechanism for cumulatively applying the policies identified by the labels within that hierarchy.

An application that is appropriately instrumented to call the val_create_context() function can supply its own validation scope to the validator library. In cases where the the application code base does not call val_create_context(), the validator library provides two additional alternatives for enabling the user to select different validator policies for different applications.

In the first option the validator library automatically attempts to construct a label from the application name and looks for a policy identified by that label in the configuration system. In the second option, the validator operator sets the value of an environment variable, which is used to communicate the desired policy scope value to the validator library. These options can be selectively enabled, thus providing for deployment scenarios ranging from a locked-down set of systems with a uniform DNSSEC validation policy to a scenario where individual users are given the flexibility to customize DNSSEC validation policy settings for their own systems or applications.

IV. API Usage Within Applications

Internet applications that make use of the Domain Name System have been developing for over twenty years. The DNS protocol has received very little updates during this period. Thus, many application developers have taken significant liberties when processing DNS error conditions based on the invalid assumption that no further changes would occur to the DNS. The result is that some applications have error handling loops that do not have proper fallback cases to default "unknown error" handling code. This situation poses problems for retrofitting DNSSEC capability into existing applications. Most importantly, it makes it very difficult for libraries offering DNS resolution support to upgrade their API to support DNSSEC because it is often unclear how many clients would break when new error codes are returned.

When DNSSEC validation fails there are at least three choices of errors that can be shown to the user. For applications that choose not to expose the details of validation failures to the user a simple "service unavailable" message is typical. This message would be indistinguishable from other unavailability events (like a network being down). Another option is to actually explain that answers were returned but they were not valid by using a "DNSSEC Validation Failure" error message. This provides more information to the end-user and their network administrator.

This section describes the work done to incorporate the DNSSEC-compliant API into two popular open-source applications: Firefox and OpenSSH. Both the difficulties in their upgrade process as well as the realized benefits are discussed.

A. DNSSEC-enabled Firefox

Mozilla's Firefox [8] is a very popular open-source web-browser. It's source code contains an immense and complex interaction of both C and Javascript. Fortunately all DNS processing within the Firefox code is funneled through one utility library. This utility library, called *libnspr* , was written to act as a buffer between operating system specific DNS libraries and the rest of the firefox code. Because it acts as a portable DNS resolving library it has also been used by other applications striving for cross-platform support. These other applications include popular applications such as OpenOffice, Thunderbird, and Evolution.

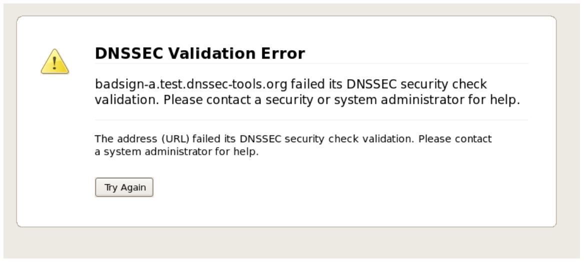
Internally to *libnspr* many of the common system-level DNS routines are called and their results are returned to the caller. Much of the wrapping code merely ensures that *libnspr* API can be used consistently across multiple platforms and that the functions can always be called in a thread-safe manner even if the underlying OS does not support thread-safe DNS lookups.

The good news with the *libnspr* implementation is that very little looked to be needed to replace the existing code with the DNSSEC-Tools' DNSSEC compliant libval API. Unfortunately, within the rest of the Firefox and other dependent code, developers making use of the *libnspr* API made the assumption that no new error codes would be returned in the future. The result is that if the *libnspr* code started returning new error messages then the web browser's code would continue into the success portion of the error checking code rather than indicating a DNS lookup failure. Multiple other portions of code within the Firefox package had to be modified in order to account for these poorly coded error handling routines.

Because other applications, as discussed above, make use of the *libnspr* API and because there are significant numbers of third-parties extensions for Firefox it is deemed unsafe to return new error codes from the existing API without a mechanism of ensuring that the caller is aware of the extended DNSSEC validation result codes. Thus a new API was introduced into *libnspr* with functions that had identical arguments to the original functions but were renamed to include *Extended* at the end of the function names. In this way callers could use the *Extended* suffixed functions when they wished to handle DNSSEC enabled return codes. The original functions still performed DNSSEC validation but would simply return an older "Host Not Found" style error code when validation failed instead of one of the extended return codes.

This solution provided both DNSSEC validation for applications using the *libnspr* API regardless of whether they were DNSSEC-aware or not while still providing detailed error status for applications that wish to migrate to becoming more DNSSEC-aware.

a) Extended Error Benefits: Once the primary Firefox code was updated to reflect new DNSSEC error conditions the error screens shown to the user were updated. Details of exactly how DNSSEC validation might have failed are not shown to the user as this is perceived as detrimental by the Mozilla team that develops Firefox. The Mozilla team has recently taken the approach to only explain to the user that a connection has failed without providing intricate details that will only confuse their average user. Figure 1 is a screen-shot of Firefox's new DNSSEC validation error screen.



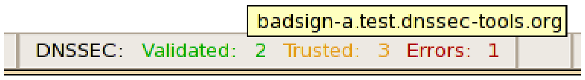
DNSSEC Validation Error

badsign-a.test.dnssec-tools.org failed its DNSSEC security check validation. Please contact a security or system administrator for help.

The address (URL) failed its DNSSEC security check validation. Please contact a system administrator for help.

Fig. 1. Firefox DNSSEC Validation Error

The DNSSEC-Tools package also contains a Firefox extension that shows the DNSSEC validation results for all the DNS lookups that were required to load a particular web page. It shows counters for the number of validated results, the number of trusted results (i.e. DNS lookups that were trusted because no validation trust anchors were available) and DNSSEC validation errors. It frequently takes many DNS requests to fully load a given web page because web pages are frequently filled with references to other sources of data, such as images, javascript source code and external cascading style sheets definitions. Figure 2 is a screen-shot of this extension in operation showing multiple look-ups when browsing the http://www.DNSSEC-Tools.org/ web site..



badsign-a.test.dnssec-tools.org

DNSSEC: Validated: 2 Trusted: 3 Errors: 1

Fig. 2. Firefox DNSSEC Display Extension

Another benefit to DNSSEC validation is actually the improved handling for when domain names do not exist. Without DNSSEC validation, if a user enters an invalid name into Firefox then the resulting error screen that firefox displays is full of potentially useful questions that help users determine what's wrong: "Did you make a mistake when typing the domain?", "Are you certain this domain address exists?", and (summarized) "Is your network connection functioning?". DNSSEC provides applications with positive proof that a domain does not exist. Because of this, the second two possible questions can be safely removed from the screen shown to the user when DNSSEC validation has proven that a domain doesn't exist. The second question can be replaced with a definitive statement: "The domain you entered does not exist". This greatly improves the simplicity of the error messages shown to the user.

B. *DNSSEC-enabled OpenSSH*

The OpenSSH [9] software suite is a very popular software suite that provides a free, open-source implementation of the Secure Shell protocol [12]. OpenSSH performs a number of DNS related activities including look-up of the hostname to connect to as well as searching for SSH Key fingerprints stored in the DNS [13].

Replacing the older DNS API used within OpenSSH with the newer DNSSEC supporting API was not a difficult task. The error handling code within OpenSSH was neither as complex as the Firefox code nor was the lookup code used in third party applications like Firefox's *libnspr* component.

A new feature not previously available to the OpenSSH implementation was also quickly added. In particular, unknown SSH keys from a remote server that can be verified through the use of the DNS key fingerprint record (*SSHFP*) can now be automatically trusted as authenticated. The original OpenSSH code could check the *SSHFP* record but could only provide guidance to the user about accepting it since the lookup of the *SSHFP* record itself was insecure. With a DNSSEC signed *SSHFP* record, however, the lookup can be completely trusted when validation succeeds and the patch provides support for auto-accepting validated SSH keys. All of this work, including adding the configuration file tokens for making auto-acceptance optional, took significantly less time than the Firefox implementation because the code base was both simpler and had a better architecture for future DNS return-code expansion.

V. Lessons Learned

A number of important discoveries were made in the process of instrumenting existing applications with a DNSSEC enabled resolution API.

Unfortunately many applications are designed with the notion that the DNS will never change and require significantly more work than simply replacing the older API with a newer one. Each application has to be analyzed to determine where DNS errors are propagating to and each point checked to ensure proper processing of newer API return codes. Some applications updated by the developers working on DNSSEC-Tools, though, were fairly trivial to update because their internal architectures were well defined and forward-thinking. These applications weren't discussed here but include *wget*, *lftp*, *sendmail*, *postfix*, *libspf* and others. The full list of applications that were made DNSSEC-capable by the DNSSEC-Tools project can be found at [14].

A positive benefit of instrumenting applications with DNSSEC is that it often reduces the vagueness of the error messages they return to definitive statements when domain names do not exist. This unexpected benefit to DNSSEC's proof of non-existence can be a great boon to helping end-users understand their usage mistakes.

Some applications that rely on DNS resource records for distributing information, such as OpenSSH, can significantly improve the security of the application when updated to use a DNSSEC enabled API.

VI. Future Work

The DNSSEC-Tools project modified only a small number of the existing open-source applications that make use of the DNS. There are many other applications that could benefit from similar changes to those performed by the project team.

The DNSSEC-Tools *libval* library is packaged with a few operating systems already, such as Fedora and Debian, but providing even wider distribution would help application writers make easy use of the DNSSEC API.

The patches that have been developed for the instrumented applications need be propagated up to the base distributions so others can more easily take advantage of the improvements. Right now, end-users would have to manually apply the patches and rebuild the components themselves.

The project team has recently begun to explore providing a system-wide "shim" library that automatically extends DNSSEC protection to all applications by replacing the normal system API with a newer one. The applications wouldn't be able to take advantage of the extended error codes but would at least be protected on a system-wide basis and be able to use a feature-rich validator policy, either on a per-application or on a system-wide basis. Additional information could still be returned to the user via a system-tray pop-up dialog box if user notification is needed for events that can't be effectively communicated through the application due to the limited error code handling available.

VII. Conclusions

Although sometimes significant work is needed to update an application so that it becomes DNSSEC aware, there is significant added benefit once the work is completed. Applications not only offer additional security features but can also offer more decisive messages to return to the end-user.

The authors believe that the benefits to in-application DNSSEC validation, particularly with respect to enabling fine-grained local control and providing better error messages to the end-user, far outweigh the costs. The distribution and management complexity of such a system is certainly higher. However, the benefit of a completely secure deployment regardless of location and the added security and usability benefits provided to the application significantly outweigh the required complexity increase.

References

[1] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034 (Standard), Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt

[2] ——, "Domain names - implementation and specification," RFC 1035 (Standard), Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033 (Proposed Standard), Mar. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4033.txt

[4] ——, "Resource Records for the DNS Security Extensions," RFC 4034 (Proposed Standard), Mar. 2005, updated by RFC 4470. [Online]. Available: http://www.ietf.org/rfc/rfc4034.txt

[5] ——, "Protocol Modifications for the DNS Security Extensions," RFC 4035 (Proposed Standard), Mar. 2005, updated by RFC 4470. [Online]. Available: http://www.ietf.org/rfc/rfc4035.txt

[6] D. Atkins and R. Austein, "Threat Analysis of the Domain Name System (DNS)," RFC 3833 (Informational), Aug. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3833.txt

[7] S. Inc., "Dnssec-tools: Dnssec software libraries and tools," http://www.dnssec-tools.org/. [Online]. Available: http://www.dnssec-tools.org/

[8] M. Corporation, "Firefox: An open-source web browser," http://www.mozilla.com/firefox. [Online]. Available: http://www.mozilla.com/firefox/

[9] C. Effort, "Openssh: a free version of the ssh connectivity tools," http://www.openssh.org/. [Online]. Available: http://www.openssh.org/

[10] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)," RFC 2845 (Proposed Standard), May 2000, updated by RFC 3645. [Online]. Available: http://www.ietf.org/rfc/rfc2845.txt

[11] R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens, "Basic Socket Interface Extensions for IPv6," RFC 3493 (Informational), Feb. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3493.txt

[12] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251 (Proposed Standard), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4251.txt

[13] J. Schlyter and W. Griffin, "Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints," RFC 4255 (Proposed Standard), Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4255.txt

[14] S. Inc., "Dnssec-tools instrumented application list," http://www.dnssec-tools.org/wiki/index.php/DNSSEC_Applications. [Online]. Available: http://www.dnssec-tools.org/wiki/index.php/DNSSEC_Applications