Vol.2016-CSEC-73 No.3
Vol.2016-IOT-33 No.3
2016/5/26

端末上で動作するDNSSEC検証及び警告システムの設計と実装

栫　邦雄[1,a)]　金　勇[2,b)]　山井　成良[1,c)]　北川　直哉[1,d)]　友石　正彦[2,e)]

概要：DNS キャッシュポイズニングは，DNS キャッシュサーバのキャッシュに偽造した情報を注入する攻撃手法である．DNS キャッシュポイズニングへの有効な対策の 1 つとして，DNSSEC が挙げられる．しかし，DNSSEC は，検証に失敗した場合に名前解決不能となったり，共用の DNS キャッシュサーバの負荷が増大したりする問題を抱えている．本論文では，上記の問題の解決方法として，ユーザの端末上で DNSSEC 検証を行うシステムを提案する．またこの手法を実装し，DNSSEC 検証に失敗した際には名前解決の結果をクライアントに返しつつ，検証失敗した旨を通知するシステムについて報告する．

キーワード：DNS，DNSSEC，DNSSEC 検証

Design and Implementation of A Client Based DNSSEC Validation and Alert System

Kunitaka Kakoi[1,a)]　Yong Jin[2,b)]　Nariyoshi Yamai[1,c)]　Naoya Kitagawa[1,d)]
Masahiko Tomoishi[2,e)]

Abstract: DNS cache poisoning is an attack that an attacker sends forged records to cache DNS server as reply to DNS query. DNSSEC protects cache DNS server from DNS cache poisoning . DNSSEC have several problems. One of the problem is that cache DNS server can't resolve FQDN in case of DNSSEC validation failure. Furthermore, the load of cache DNS server increases for DNSSEC validation. In this paper, in order to resolve these problems, we propose design of client based DNSSEC validation and implement a system that notifies users that DNSSEC validation is failed.

Keywords: DNS, DNSSEC, DNSSEC validation

1. はじめに

現在，インターネットは広く普及しており，我々の生活に必要不可欠と言っても過言ではない程に重要である．インターネットではネットワークに接続しているコンピュータを識別するために，各ホストには固有の IP アドレスが付けられ，通信を行う際にはその IP アドレスが使用される．しかし，IP アドレスは単なる数字の列であり人間にとっては覚えにくいため，人間が覚えやすいように複数の短い英字を「.」によって繋いだ構造となっているドメイン名やホスト名を別名として用いている．DNS(Domain Name System) とは，インターネット上でドメイン名やホスト名と IP アドレスの対応関係を効率よく管理するシステムで，インターネットの重要な基盤技術の 1 つである．

DNS では，プロトコルの仕様上，送られてきた DNS 情報は基本的には信用して受け取る．そのため，攻撃者が送

[1] 東京農工大学
Tokyo University of Agriculture and Technology, Koganei, Tokyo 184-8588, Japan
[2] 東京工業大学
Tokyo Institute of Technology, Meguroku, Tokyo 152-8550, Japan
[a)] s163901w@st.go.tuat.ac.jp
[b)] yongj@gsic.titech.ac.jp
[c)] nyamai@cc.tuat.ac.jp
[d)] nakit@cc.tuat.ac.jp
[e)] tomoishi@noc.titech.ac.jp

信した悪意のある偽の DNS 情報も受け取ってしまう可能性がある．DNS のプロトコルの脆弱性に目を付けた攻撃の 1 つに DNS キャッシュポイズニングがある．DNS キャッシュポイズニングは以前から知られていたが，現実的な成功率は低く，簡単には成功しないと考えられていた．しかし，2008 年に Dan Kaminsky によって発表されたカミンスキー型攻撃は，従来の手法より攻撃の成功率を飛躍的に高めることができ，その危険性は現実的なものとなった．そのため，カミンスキー型攻撃の成功率の高さは大きな問題として扱われた．そこで，今までは無条件に信用して受け取っていた DNS 情報の正当性を検証できるような仕組みが必要となり，注目されたのが DNSSEC である．

DNSSEC は，1999 年に初めて標準化されたが [1]，2005 年にそれを改良して再び標準化された [2][3][4]．DNSSEC は標準化されてから 10 年以上経過した今現在でも決して広く普及していない状況が続いている．DNSSEC が普及しない主な原因としては運用管理の複雑さが挙げられる．DNSSEC では，DNS 情報の正当性を検証するために，公開鍵暗号方式による電子署名技術を名前解決の処理に応用しており，権威 DNS サーバは署名検証に使用する鍵を適切に管理する必要がある．また，セキュリティ上の理由から鍵や署名の更新を定期的に行う必要がある．鍵や署名の更新に失敗した場合，その権威 DNS サーバの管理するゾーンについて全て DNSSEC 検証に失敗するため，名前解決が不可能となり，DNS を利用するサービスが使えなくなる可能性がある．また，DNSSEC 検証は DNS キャッシュサーバが行うため，その際に DNS キャッシュサーバが消費する CPU やメモリのリソースが増加する [5][6]．これにより，DNS キャッシュサーバの本来の機能である名前解決の性能が低下する問題がある．

これらの問題を解決する手段として，本論文では，通常はキャッシュサーバで行われる DNSSEC 検証を，ユーザの端末上で行うシステムを提案する．第 2 章で本システムの関連研究について述べた後，第 3 章で DNSSEC の課題とその解決策として提示する本システムの基本設計について述べ，第 4 章では本システムの実装方法について述べ，その動作確認を行い，第 5 章にてまとめる．

2. DNS キャッシュポイズニングと DNSSEC

2.1 DNS キャッシュポイズニング

DNS では，キャッシュという仕組みを利用して，DNS キャッシュサーバの負荷の軽減や高速化を図っている．DNS キャッシュポイズニングとは，悪意のある攻撃者が DNS キャッシュサーバに偽の DNS 情報を含む応答を送り込み，その偽の DNS 情報を DNS キャッシュサーバにキャッシュとして保持させ，クライアントに偽の DNS 情報を返させる攻撃手法である．

2.1.1 DNS キャッシュポイズニングの一般的な手法

以下に DNS キャッシュポイズニングの一般的な手法の攻撃手順を示す (図 1)．

(1) ユーザは DNS キャッシュサーバに「www.example.com」の名前解決の問い合わせを行う．
(2) DNS キャッシュサーバは example.com の権威 DNS サーバに名前解決の問い合わせを行う．
(3) 攻撃者は，権威 DNS サーバからの正当な応答が返って来る前に，偽の DNS 情報を含む応答を DNS キャッシュサーバに送り込む．
(4) DNS キャッシュサーバは偽の DNS 情報をキャッシュして，ユーザに偽の DNS 情報を応答する．
(5) ユーザは知らない内に偽のサイトに誘導される．

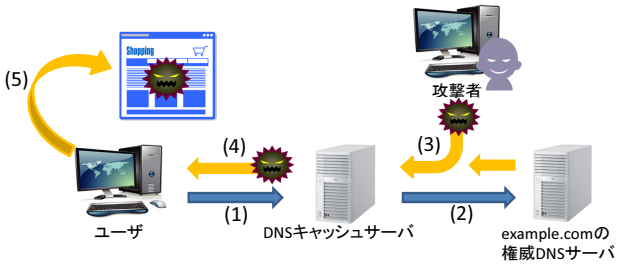

図 1　DNS キャッシュポイズニング

上記のような方法で DNS キャッシュポイズニングを行う場合，攻撃者は DNS キャッシュサーバに対して偽造した応答を正当な応答よりも早く送り込む必要がある．応答を偽造するためには，送信元・送信先 IP アドレス，送信先ポート番号，問い合わせ先ポート番号，問い合わせ先ドメイン・クラス・タイプ，問い合わせ ID を正当な応答と一致させる必要がある．問い合わせ ID の取り得る値は $2^{16} = 65536$ 通りであるため，問い合わせ ID は総当たりで一致させることが可能である [7]．DNS キャッシュポイズニングの対策として，ソースポートランダマイゼーション [8][9][10] が挙げられる．ソースポートランダマイゼーションは送信先ポート番号をランダムに変化させて，偽造された応答が到達するポート番号を一致させにくくし，DNS キャッシュポイズニングの成功率を低下させる．また，権威 DNS サーバから得られたリソースレコードは TTL に従って，DNS キャッシュサーバにキャッシュとして保持され続ける．そのキャッシュが保持されている間は，DNS キャッシュサーバは権威 DNS サーバにそのリソースレコードに対する問い合わせを行わない．つまり，リソースレコードのキャッシュが保持されている間は，そのリソースレコードに関する応答を偽造して送り込めない．したがって，TTL を十分に長く設定することにより，DNS キャッシュポイズニングの成功率を低下させることができると考えられている．

2.1.2 カミンスキー型攻撃

TTL を長くすることにより，DNS キャッシュポイズニングのリスクを減少させられると考えられていた．しかし，2008 年 8 月に Dan Kaminsky によって新たな手法の DNS キャッシュポイズニングが発見された [11]．この新たに発見された攻撃手法は TTL の長さに関係なく連続的に攻撃できる．以下にカミンスキー型攻撃の攻撃手順を示す (図 2)．

(1) 攻撃者は DNS キャッシュサーバに攻撃したいドメイン名と同じドメイン内の存在しないドメイン名を問い合わせる．例えば「www.example.com」のドメイン名に対して攻撃する場合，「<ランダム文字列>.example.com」について問い合わせる．
(2) 「<ランダム文字列>.example.com」は存在しないドメイン名であるため，DNS キャッシュサーバは example.com の権威 DNS サーバに問い合わせを行う．
(3) example.com の権威 DNS サーバは「<ランダム文字列>.example.com というドメイン名は存在しない (NXDOMAIN)」という応答を DNS キャッシュサーバに返す．
(4) 攻撃者は権威 DNS サーバからの正当な応答よりも先に送信元 IP アドレスを詐称した偽の応答を DNS キャッシュサーバに送りつける．例えば，偽の応答には「そのドメイン名は他の権威 DNS サーバに問い合わせを行え，そのサーバ名は「www.example.com」で IP アドレスは 192.0.2.1(192.0.2.1 は攻撃者が用意した偽の IP アドレス)」といった情報を付加する．
(5) 偽の DNS 情報をキャッシュさせることが成功するまで，(1) から (4) の手順を何度も繰り返す．

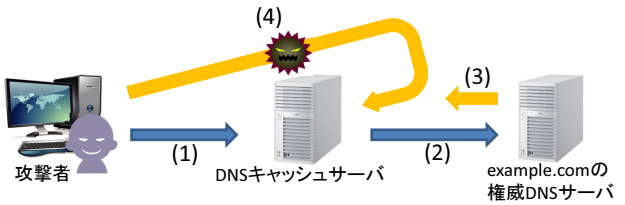

図 2　カミンスキー型攻撃

上記のように，カミンスキー型攻撃は，問い合わせ先ドメイン名に対する偽造リソースレコードを直接送り込むのではなく，問い合わせへの応答に付加した DNS 情報を用いることにより，DNS キャッシュサーバに偽の DNS 情報をキャッシュさせることができる．このように，カミンスキー型攻撃では攻撃に用いるドメイン名をランダムに変えることにより，TTL で示されたキャッシュの保持時間を待たずに連続攻撃が可能となる．

2.2 DNSSEC

DNSSEC は，公開鍵暗号方式の電子署名技術を基に，DNS キャッシュサーバによる問い合わせへの応答が正当であるかを検証するために利用される．応答がゾーンの管理者が登録したとおりの内容で，通信途中で書き換えられていない場合，その応答は正当である．前節で述べた DNS キャッシュポイズニングは，正当ではない DNS サーバから偽造された応答パケットを送ることにより攻撃を行う．したがって，DNSSEC を用いることにより DNS キャッシュポイズニングを防ぐことができる．

2.2.1 電子署名

DNSSEC は公開鍵暗号方式の電子署名技術を利用している．公開鍵暗号方式とは，データの暗号化に作成者が保持している秘密鍵を用いて，データの復号には予め外部に公開している公開鍵を用いる方式である．公開鍵暗号方式の特徴は，秘密鍵で暗号化したデータはペアとなる公開鍵でしか復号できないこと，一方の鍵からペアとなる他方の鍵を推測することが大変困難なことである．以下に公開鍵暗号方式の電子署名の利用手順を説明する (図 3)．

(1) データの送信者はハッシュ関数を用いて送信するデータのハッシュ値を計算する．
(2) 得られたハッシュ値を秘密鍵を用いて暗号化し，署名を作成する．
(3) 送信者はデータに署名を添付して受信者に送信する．
(4) データと署名を受け取った受信者は送信者が用いたものと同じハッシュ関数を用いてデータのハッシュ値を計算する．
(5) データに添付された署名を予め入手しておいた公開鍵で復号してハッシュ値を得る．
(6) (4) と (5) で得られたハッシュ値を比較して，一致するかを検証する．

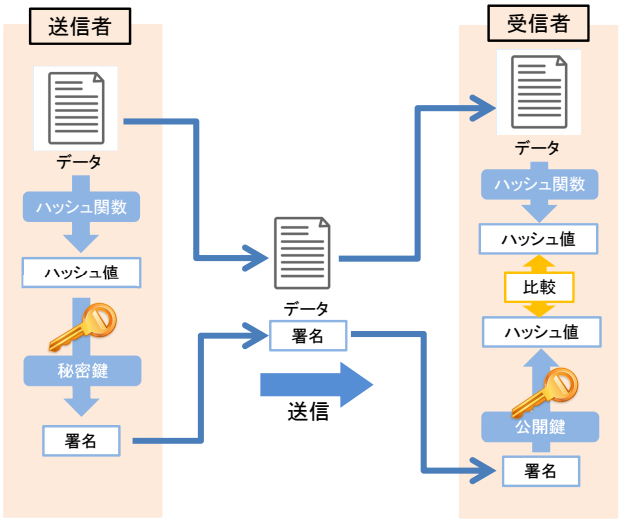

図 3　電子署名による検証

2.2.2 DNSSEC の仕組み

DNSSEC では前節で述べた公開鍵暗号方式の電子署名を用いて DNS 応答の正当性を検証する (図 4)．権威 DNS サーバは自身の管理するリソースレコードについてハッシュ関数を用いてハッシュ値を計算して，得られたハッシュ値を秘密鍵で暗号化して RRSIG を作成する．RRSIG とはリソースレコードに対する署名のことを指す．権威 DNS サーバは DNS キャッシュサーバからの問い合わせを受け取ると，リソースレコードとその RRSIG を応答として DNS キャッシュサーバに返す．権威 DNS サーバからの応答を受け取った DNS キャッシュサーバはハッシュ関数を用いてリソースレコードからハッシュ値を計算する．また，RRSIG を DNSKEY を用いて復号してハッシュ値を求める．DNSKEY はリソースレコードを署名する秘密鍵に対応する公開鍵である．リソースレコードから計算したハッシュ値と RRSIG を復号して得たハッシュ値が一致した場合，DNSSEC 検証に成功して，権威 DNS サーバからの応答が正当であると分かる．

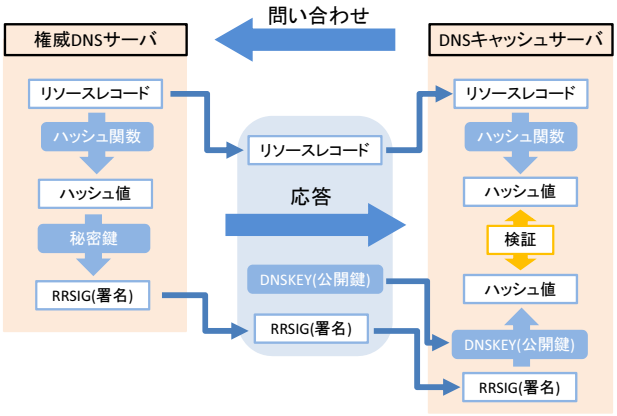

図 4　DNSSEC の仕組み

2.2.3 信頼の連鎖

前節で述べた DNSSEC 検証手法は，送信されてきた DNSKEY が正当であることが前提である．しかし，単に RRSIG と DNSKEY を受け取っただけでは，それらが確かに正当であるかの判断ができないため，攻撃者が RRSIG と DNSKEY の両方を偽装して送った場合には意味をなさない．したがって，DNSSEC では，送信されてきた DNSKEY の正当性を証明するために，「信頼の連鎖」と呼ばれる仕組みを採用している．2005 年に標準化された現在の DNSSEC では，ゾーン署名用の鍵 (ZSK:Zone Signing Key) と信頼の連鎖を構築するための鍵 (KSK:Key Signing Key) の 2 種類の鍵を利用して運用されている．

図 5 は信頼の連鎖の仕組みを表している．権威 DNS サーバは予めその上位の権威 DNS サーバに KSK 公開鍵のハッシュ値を DS として送信する．上位の権威 DNS サーバは送られてきた DS が正当かを検証し，自身の ZSK 秘密鍵で署名を行い公開する．権威 DNS サーバは，KSK 秘密鍵で ZSK 公開鍵の署名と ZSK 秘密鍵でリソースレコードの署名を行う．そして，権威 DNS サーバはキャッシュ DNS サーバからの問い合わせを受け取ると，リソースレコードとその RRSIG，ZSK 公開鍵とその RRSIG，KSK 公開鍵を応答として返す．DNS キャッシュサーバは応答を受け取ると予め入手してその署名を検証が成された DS と KSK 公開鍵のハッシュ値が一致するか検証する．そして，その KSK 公開鍵で ZSK 公開鍵の RRSIG を復号して，その値と ZSK 公開鍵のハッシュ値が一致するか検証する．最後に，その ZSK 公開鍵でリソースレコードの RRSIG を復号して，その値とリソースレコードのハッシュ値が一致するかを検証する．

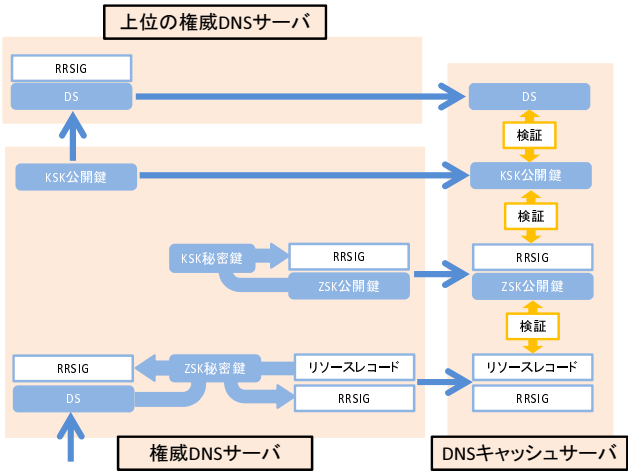

図 5　信頼の連鎖

上位の権威 DNS サーバに登録されている DS と権威 DNS サーバからの応答に含まれる DNSKEY が一致しているかを検証して，攻撃者が RRSIG と DNSKEY の両方を偽造した攻撃を防ぐ．

そして，上位の権威 DNS サーバも同様に，さらにその上位の権威 DNS サーバに DS を登録して，信頼の連鎖を構築する．また，信頼の連鎖を構築する際の起点となる情報はトラストアンカーと呼ばれる．DNSSEC を運用する際は，トラストアンカーとしてルートサーバの KSK 公開鍵または DS を予め DNS キャッシュサーバに登録しておく．

2.2.4 DNSSEC の課題

DNSSEC は DNS キャッシュポイズニングへの対策としてはきわめて有効で，DNSSEC の普及により DNS キャッシュポイズニングの被害をなくすことができる．しかし，実際に運用するにあたり，DNSSEC にはいくつかの課題がある．

- 2.2.1 節で述べたように，DNSSEC は公開鍵暗号方式の電子署名技術を利用しており，公開鍵暗号方式はセキュリティ上の理由から定期的に鍵や署名の更新を行

う必要がある．しかし，権威 DNS サーバが鍵の更新や署名の作成に失敗した場合，DNSSEC 検証に失敗し，名前解決不能となる．

- DNSSEC 検証は DNS キャッシュサーバで行われるため，名前解決の処理に加えて，DNSSEC 検証も行う必要がある．したがって，DNS キャッシュサーバの負荷が増大して，名前解決処理の性能が低下する可能性がある [12]．
- DNSSEC 検証は，DNS キャッシュサーバで行われるため，DNS キャッシュサーバからクライアントへの応答を偽造して，クライアントに偽の DNS 情報を送りつける攻撃を防ぐことができない．

3. DNSSEC 検証システムの設計

3.1 本システムの概要

本システムではユーザの端末上で DNSSEC 検証を行うシステムの実装を行った．端末上で DNSSEC 検証を行うため，利用している DNS キャッシュサーバが DNSSEC 検証に対応していない場合にも，本システムにより，ユーザは DNSSEC を利用することができる．本システムは DNSSEC 検証を DNS キャッシュサーバではなくユーザの端末上で行うため，DNS キャッシュサーバが検証を行う必要がなく，DNS キャッシュサーバの負荷を軽減させることが可能である．また，DNS キャッシュサーバからクライアント間で応答が偽造された場合，ユーザの端末上で DNSSEC 検証を行うため，偽造された応答の受信を防ぐことができる．本システムでは，DNSSEC 検証に失敗した場合に応答として「SERVFAIL」を返すのではなく，警告画面をポップアップで表示させ，ユーザに注意を促しつつ，名前解決の結果もユーザに送信する．これにより，権威 DNS サーバの鍵の更新失敗で DNSSEC 検証に失敗した場合にも，ユーザは名前解決の結果を知ることができ，偽造された応答の送信により DNSSEC 検証に失敗した場合にも，警告画面が表示してユーザに注意を促すことができる．

3.2 システムの動作設計

本システムにおけるプロクシおよびクライアントの動作を以下の図 6 に示す．

(1) スタブリゾルバは DNS 問い合わせパケットをプロクシに送る．プロクシは受け取った DNS 問い合わせパケットを DNS キャッシュサーバ 1 にフォワードする．
(2) プロクシは DNS キャッシュサーバ 1 からの応答を受け取ると，応答パケットの CD フラグを読みとり，この応答に対する DNSSEC 検証を行うかを判断する．CD=1 の場合はそのまま応答パケットをスタブリゾルバに送る．CD=0 の場合は，応答パケットから「Question Section」の情報を読みとり，その情報を DNSSEC 検証モジュールの引数として利用する．
(3) DNSSEC 検証モジュールは，クライアントのローカル内にある DNS キャッシュサーバ 2 を通して DNSSEC 検証に必要なリソースレコードを収集して，DNNSEC 検証を行う．
(4) DNSSEC 検証の結果，ドメインが DNSSEC に対応していない場合，応答パケットをそのままスタブリゾルバに送信する．
(5) DNSSEC 検証の成功した場合，検証時に得られた DNS 情報から応答パケットを作成して，それをスタブリゾルバに送信する．
(6) DNSSEC 検証に失敗した場合，検証時に得られた DNS 情報から応答パケットを作成し，それをスタブリゾルバに送信した後にユーザに注意を促すためのポップアップを表示させる．

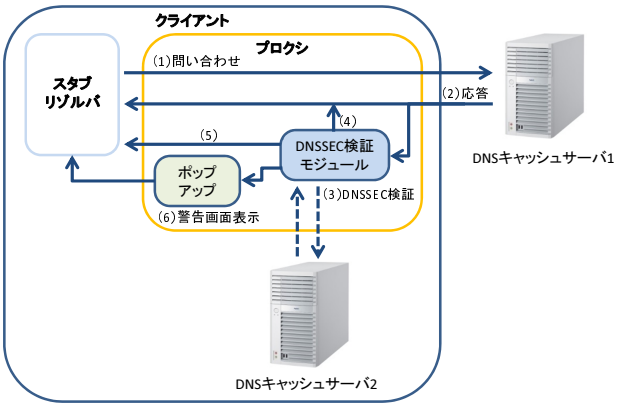

図 6 本システムの動作設計

4. DNSSEC 検証システムの実装と動作確認

4.1 システムの実装

3.2 節の設計を基に，Perl 言語を用いてシステムの実装を行った．本システムでは DNSSEC 検証を行うため，CPAN 上にある Perl モジュール Net::DNS::SEC::Validator[13] を使用した．Net::DNS::SEC ::Validator は livbal(3) という DNS リゾルバライブラリにより得られる機能を実装または出力する．DNS 問い合わせパケットを既存の DNS キャッシュサーバにフォワードして，返ってきた応答パケットを解析するプロクシを実装するために，Net::DNSServer::Proxy[14] を基に作成した Net::DNSServer::Proxy2 および，Net::DNSServer[15] を基に作成した Net::DNSServer2 を使用した．また，これらをすべて Cygwin 上で実装することにより，ユーザの端末上で動作するようにした．Net::DNSServer::Proxy は，Net::DNSServer を利用して，スタバリゾルバから送られてきた DNS 問い合わせパケットを指定した DNS キャッシュサーバにフォワードし，返ってきた応答をスタブリゾルバ

に返すプロクシを実装したプログラムである．本システムで使用した Net::DNSServer2 は，Net::DNSServer の DNS キャッシュサーバから返ってきた応答パケットをスタブリゾルバに送るという処理の間に，DNSSEC 検証を行い，その結果により，スタブリゾルバに返す応答パケットを改変したり，ポップアップを表示させたりするといった処理を追加したプログラムである．本システムでは DNSSEC 検証機能として，Net::DNS::SEC::Validator のメソッドである resolve_and_check（）を利用した．resolve_and_check（）は<name>と<class>と<type>と<flags>を引数に取り，DNS 問い合わせに関係するすべての応答や証明を複雑なデータ構造体で返す．<name>で問い合わせ先ドメイン名または IP アドレス，<class>で問い合わせるリソースレコードのクラス，<type>で問い合わせるリソースレコードのタイプを指定する．<flags>は DNSSEC 検証の制御フラグである．

resolve_and_check（）を実行すると，問い合わせに対する応答に含まれているリソースレコードの DNSSEC 検証の結果として，表 1 内の値を status code として得る．

応答の署名検証に失敗するか，認証の連鎖において不在証明 [16] の検証ができない場合，VAL_BOGUS が返される．認証の連鎖が登録されたトラストアンカーに至らない場合，VAL_NOTRUST が返される．本システムでは，status code が VAL_BOGUS(1) または VAL_NOTRUST(4) のときに応答の DNSSEC 検証に失敗したと判断する．

表 1　DNSSEC Validation Status Codes

valStatus	valStatusStr
1	VAL_BOGUS
2	VAL_DNS_ERROR
3	VAL_INDETERMINATE
4	VAL_NOTRUST
128	VAL_SUCCESS
133	VAL_NONEXISTENT_NAME
134	VAL_NONEXISTENT_TYPE
135	VAL_NONEXISTENT_NAME_NOCHAIN
136	VAL_NONEXISTENT_TYPE_NOCHAIN
137	VAL_PINSECURE
138	VAL_PINSECURE_UNTRUSTED
139	VAL_BARE_RRSIG
140	VAL_IGNORE_VALIDATION
141	VAL_UNTRUSTED_ZONE
142	VAL_OOB_ANSWER
143	VAL_TRUSTED_ANSWER
144	VAL_VALIDATED_ANSWER
145	VAL_UNTRUSTED_ANSWER

また，リソースレコードの DNSSEC 検証に成功すると VAL_SUCCESS が返される．ドメイン名の不在証明の DNSSEC 検証に成功すると VAL_NONEXISTENT_NAME が返され，指定されたリソースレコードタイプの不在証明の DNSSEC 検証に成功すると VAL_NONEXISTENT_TYPE が返される．本システムでは，status code が VAL_SUCCESS(128)，VAL_NONEXISTENT_NAME(133) または VAL_NONEXISTENT_TYPE(134) のときに応答の DNSSEC 検証に成功したと判断する．

Net::DNSServer2 における，DNS キャッシュサーバからの応答パケットを解析して，DNSSEC 検証を行い，スタブリゾルバに応答パケットを返す部分の処理フローは図 7 のようになる．

(1) キャッシュ DNS サーバから応答パケットを受け取る．
(2) 応答パケットの Question Section から問い合わせたドメイン名，クラス，タイプを取り出す．
(3) 応答パケットのヘッダーの CD フラグを確認する．CD フラグは，クライアントがサーバに応答の DNSSEC 検証を実行しないことを要求するときに設定する．よって，CD=1 の場合，DNSSEC 検証を行わないため，受け取った応答パケットをそのままスタブリゾルバに返す．CD=0 の場合，resolve_and_check() を実行する．
(4) Question Section から取り出したドメイン名とクラスとタイプをそれぞれ<name>と<class>と<type>，<flags>は VAL_QUERY_AC_DETAIL とし，resolve_and_check() を実行して，その結果を得る．
(5) 応答パケットの Answer Section 内に存在するリソースレコードについての status code を取り出し，$status とする．
(6) $status が VAL_NOTRUST(4) または VAL_BOGUS(1) の場合，DNSSEC 検証に失敗したと判断する．応答パケットの Answer Section 内のリソースレコードをすべて削除し，resolve_and_check() を実行した際に得られたリソースレコードを応答パケットの Answer Section に追加して，改変した応答パケットをスタブリゾルバに返す．その後，exec() を用いて，警告画面を表示させるプログラム「dnssec_untrust.exe」を実行する．
(7) $status が VAL_SUCCESS(128) または VAL_NONEXISTENT_NAME(133) または VAL_NONEXISTENT_TYPE(134) の場合，DNSSEC 検証に成功したと判断する．AD フラグを 1 にした応答パケットの Answer Section 内のリソースレコードをすべて削除して，resolve_and_check() を実行した際に得られたリソースレコードを応答パケットの Answer Section 内に追加して，改変した応答パケットをスタブリゾルバに返す．
(8) $status が (6), (7) 以外の場合，受け取った応答パケットをそのままスタブリゾルバに返す．

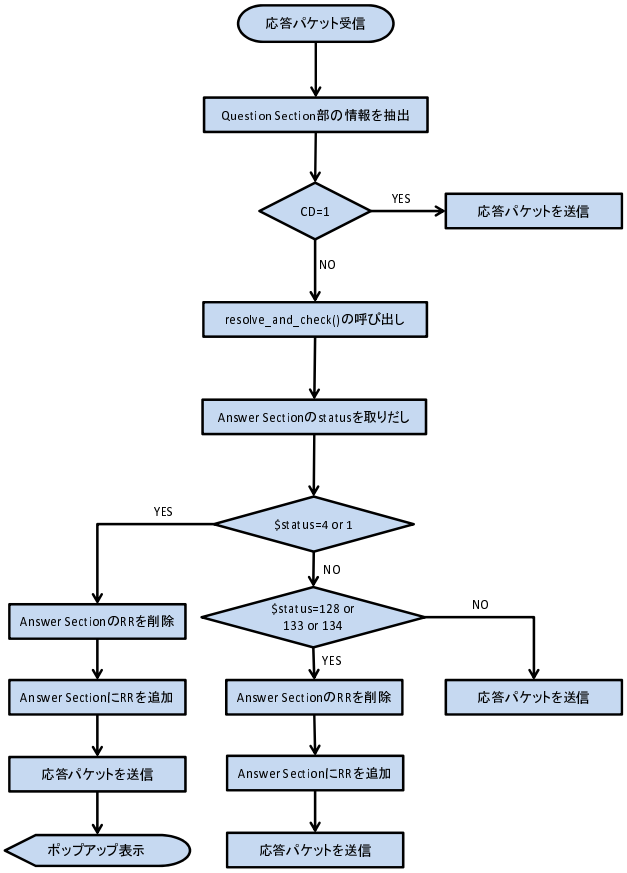

図 7　Net::DNSServer2 のフローチャート

4.2　システムの動作確認

本システムを研究室の PC(IP アドレスは 165.93.176.94) 上に実装して，実際に動作させた．resolve_and_check() の実行時に DNSSEC 検証に必要なリソースレコードを問い合わせる DNS キャッシュサーバとして，ローカル内に立てた自前の DNS キャッシュサーバ (BIND9.10) を利用した．DNS 問い合わせパケットのフォワード先の DNS キャッシュサーバ Google Public DNS(8.8.8.8) を利用した．また，PC のネットワーク設定において，本システムに DNS パケットをフォワードするため，DNS キャッシュサーバのアドレスをループバックアドレス (127.0.0.1) に設定した．

コマンドプロンプト上で「dig www.google.co.jp」を実行した場合，「www.google.co.jp」は DNSSEC に対応していないため，本システムで DNSSEC の検証を行った後，DNS キャッシュサーバから受け取った応答パケットはスタブリゾルバにそのまま返された．

また，「dig jprs.jp」を実行すると，「jprs.jp」は DNSSEC に対応しているため，本システムでの DNSSEC 検証に成功した．よって，Answer Section のリソースレコードを検証の際に得たリソースレコードに書き換えて，AD フラグを 1 にした応答パケットがスタブリゾルバに返された．

次に本システムを利用せずに Google Public DNS(8.8.8.8) に名前解決の問い合わせを行うため，コマンドプロンプト上で「dig validation-error.dnslab.jp +dnssec @8.8.8.8」を実行すると，図 8 の様な結果を得た．「validation-error.dnslab.jp」は DNSSEC に対応しているが，A レコードの署名が意図的に改竄されているため，応答の署名の検証に失敗する．したがって，Google Public DNS(8.8.8.8) は DNSSEC 検証を行い，応答は「SERVFAIL」となった．本システムを利用した場合として，コマンドプロンプト上で「dig validation-error.dnslab.jp +dnssec 」を実行すると，図 9 の様な結果を得た．本システムを利用することにより，図 8 のように応答が「SERVFAIL」とならずに，名前解決の結果が得ることができた．また，DNSSEC 検証に失敗したことをユーザに警告するためのポップアップ（図 10）が表示された．

```
$ dig validation-error.dnslab.jp @8.8.8.8

; <<>> DiG 9.9.8-P2 <<>> validation-error.dnslab.jp @8.8.8.8
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 30751
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;validation-error.dnslab.jp.    IN      A

;; Query time: 3690 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Feb 04 05:01:18 Asi 2016
;; MSG SIZE  rcvd: 55
```

図 8　「dig validation-error.dnslab.jp @8.8.8.8」の実行結果

```
$ dig validation-error.dnslab.jp

; <<>> DiG 9.9.8-P2 <<>> validation-error.dnslab.jp
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52085
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;validation-error.dnslab.jp.    IN      A

;; ANSWER SECTION:
validation-error.dnslab.jp. 60  IN      A       127.0.0.2

;; Query time: 1050 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Jan 19 05:33:00 Asi 2016
;; MSG SIZE  rcvd: 71
```

図 9　「dig validation-error.dnslab.jp 」の実行結果

5.　むすび

本論文では，ユーザの端末上で動作して，DNS キャッシュサーバからの応答に対して DNSSEC 検証を行い，DNSSEC 検証に失敗した場合にはユーザに検証の失敗を通知するシステムについて述べた．DNS キャッシュサーバは DNSSEC

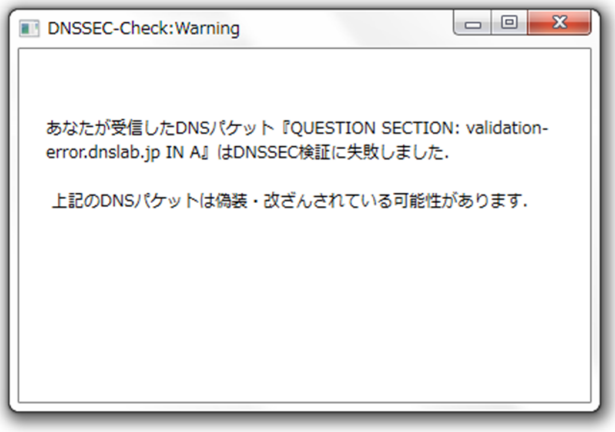

図 10 「validation-error.dnslab.jp」に対するポップアップ通知

検証に失敗した際，それが鍵の更新ミスなどの DNSSEC 運用面でのミスが原因であっても，名前解決不能となり，応答として「SERVFAIL」を返す．しかし，本システムでは，DNSSEC 検証に失敗した場合，名前解決不能とせずに名前解決の結果を返して，DNSSEC 検証の失敗をポップアップを用いてユーザに通知して注意を喚起する．これにより，DNSSEC 運用面でのミスによって DNSSEC 検証に失敗した場合でも，ユーザは名前解決の結果を受信できる．本システムにより，DNSSEC 検証を行わない DNS キャッシュサーバを利用する場合にも，ユーザは DNSSEC による応答パケットの正当性の保証を受けられる．また，ユーザの端末上で DNSSEC 検証を行うため，DNS キャッシュサーバの負荷の軽減に繋がり，DNS キャッシュサーバからクライアントへの応答を偽造する攻撃の防止が可能となる．このように，本システムは，DNSSEC 運用時のいくつかの懸念を解消できる．

本論文では，本システムを設計・実装して実際に動作確認をするだけに止まっているが，本システムを利用時の DNS パケットの通信量や名前解決の処理時間を調査する性能評価の実施を今後の課題とする．また，検証の高速化も本システムの課題として挙げられる．本システムは基本的には DNS キャッシュサーバから応答が返って来るたびに毎回 DNSSEC 検証を行う．よって，プロクシにおける DNSSEC 検証の処理時間を短縮するため，キャッシュ機能の実装が考えられる．プロクシで検証を行って，DNSSEC に対応していないと判断されたドメインはキャッシュに登録して，次回からそのドメインからの応答は DNSSEC 検証を行わないようにして，無駄な DNSSEC 検証処理を省いて処理時間を短縮できると考えられる．

参考文献

[1] Eastlake, D. E. et al.: Domain name system security extensions (1999).
[2] Arends, R., Austein, R., Larson, M., Massey, D. and Rose, S.: DNS security introduction and requirements (2005).
[3] Arends, R., Austein, R., Larson, M., Massey, D. and Rose, S.: Resource records for the DNS security extensions (2005).
[4] Arends, R., Austein, R., Larson, M., Massey, D. and Rose, S.: Protocol modifications for the DNS security extensions (2005).
[5] 副島裕司，若杉泰輔，島村祐一， 平野衡， 岡英一：DNS キャッシュサーバにおける DNSSEC 性能評価，電子情報通信学会技術研究報告. IN, 情報ネットワーク， Vol. 108, No. 426, pp. 37–42 (2009).
[6] 若杉泰輔，副島裕司，島村祐一， 岡英一：署名パターンに着目した DNS キャッシュサーバの DNSSEC 性能評価，電子情報通信学会技術研究報告. IN, 情報ネットワーク， Vol. 109, No. 119, pp. 61–66 (2009).
[7] Stewart, J.: DNS cache poisoning–the next generation (2003).
[8] Larsen, M. and Gont, F.: Recommendations for transport-protocol port randomization (2011).
[9] Dougherty, C. R.: CERT Vulnerability Note VU 800113: Multiple DNS implementations vulnerable to cache poisoning (2008).
[10] Klein, A.: BIND 9 DNS cache poisoning (2007).
[11] CERT, U.: Vulnerability Note VU# 800113: Multiple DNS implementations vulnerable to cache poisoning (2008).
[12] JPRS: DNSSEC 技術実験報告書 機能・性能確認編, 入手先 ⟨http://jprs.jp/dnssec/doc/DNSSEC-testbed-report-fpv1.0.pdf⟩ （参照 2016-2-25）.
[13] CPAN: CPAN:Net::DNS::SEC::Validator, available from ⟨http://search.cpan.org/ gsm/Net-DNS-SEC-Validator-1.31/Validator.pm⟩ (accessed 2016-2-25).
[14] CPAN: CPAN:Net::DNSServer::Proxy, available from ⟨http://search.cpan.org/ bbb/Net-DNSServer-0.11/lib/Net/DNSServer/Proxy.pm⟩ (accessed 2016-2-25).
[15] CPAN: CPAN:Net::DNSServer, available from ⟨http://search.cpan.org/ bbb/Net-DNSServer-0.11/lib/Net/DNSServer.pm⟩ (accessed 2016-2-25).
[16] Blacka, D., Laurie, B., Sisson, G. and Arends, R.: DNS security (DNSSEC) hashed authenticated denial of existence (2008).