An On-Demand Defense Scheme Against DNS Cache Poisoning Attacks

Zheng Wang[1(✉)], Shui Yu[2], and Scott Rose[1]

[1] National Institute of Standards and Technology, Gaithersburg, MD 20899, USA
zhengwang98@gmail.com, scott.rose@nist.gov
[2] School of Information Technology, Deakin University, Burwood, VIC 3125, Australia
syu@deakin.edu.au

Abstract. The threats of caching poisoning attacks largely stimulate the deployment of DNSSEC. Being a strong but demanding cryptographical defense, DNSSEC has its universal adoption predicted to go through a lengthy transition. Thus the DNSSEC practitioners call for a secure yet lightweight solution to speed up DNSSEC deployment while offering an acceptable DNSSEC-like defense. This paper proposes a new On-Demand Defense (ODD) scheme against cache poisoning attacks, still using but lightly using DNSSEC. In the solution, DNS operates in DNSSEC-oblivious mode unless a potential attack is detected and triggers a switch to DNSSEC-aware mode. The modeling checking results demonstrate that only a small DNSSEC query load is needed by the ODD scheme to ensure a small enough cache poisoning success rate.

Keywords: DNS Security Extensions · DNS cache poisoning
Model checking · Query load · Success rate

1 Introduction

The Domain Name System (DNS) is todayąś's largest name resolution system in use. As a critical component in networking infrastructure, the DNS is becoming an increasingly lucrative target for adversaries. However, the early design of DNS did not pay sufficient attention to its security in 1980s. One major progress on securing DNS is DNS Security Extensions (DNSSEC) [1,2] as a set of core specifications agreed by IETF in 2005. DNSSEC provides security capabilities by digitally signing DNS data using public-key cryptography.

DNSSEC deployment was essentially motivated as a response to the Kaminsky vulnerability [3] which allows attackers to inject bogus DNS responses with a considerable success rate. While DNSSEC convincingly secures the DNS from the Kaminsky attacks, the concerns over DNSSEC overheads have posed big obstacles to its adoption. The impacts of DNSSEC on DNS performance are multi-facet:

X. Lin et al. (Eds.): SecureComm 2017, LNICST 238, pp. 793–807, 2018.
https://doi.org/10.1007/978-3-319-78813-5_43

- The number of queries required by DNSSEC-aware resolution is amplified [4].
- The average packet size generated by DNSSEC is enlarged [6].
- The query processing cost at both authoritative servers and recursive servers is increased by DNSSEC [5,7].

Hence DNSSEC deployment commonly means heavy investments, great efforts, and stability risks for DNS operators and DNS service providers. Such concerns may best explain the fact that the universal DNSSEC adoption is still very far from completion despite of the prominent demands for DNS security.

One promising way of promoting DNSSEC deployment is to limit DNSSEC overheads in order to make DNSSEC more affordable for DNS operators and DNS service providers. Perhaps the most obvious way to cut DNSSEC costs is to limit DNSSEC transactions between authoritative servers and recursive servers. That is, minimizing DNSSEC-enabled queries issued from recursive servers and processed by authoritative servers. Admittedly, the tradeoff between DNSSEC usage and security capability always stands. Nevertheless, an efficient use of DNSSEC, hopefully, mitigates DNS servers loads while offering an acceptable DNSSEC-like defense.

The defense proposed in this paper, namely ODD (On-Demand Defense), basically secures recursive resolvers against any off-path cache poisoning attacks. It still uses but lightly uses DNSSEC in a bid to lower its DNSSEC overheads. ODD makes full use of the detection capability of recursive resolvers to take up DNSSEC whenever needed. The rest of this paper is organized as follows. Related work is presented in Sect. 2. The ODD scheme is elaborated in Sect. 3. In Sect. 4, we present the performance analysis of the ODD scheme. Section 5 evaluates the ODD scheme through model checking. Finally, Sect. 6 concludes the paper.

2 Related Work

Before or in parallel with the DNSSEC rollout, there have been some proposals attempting to address the DNS cache poisoning risks in a light-weight way. As a non-DNSSEC solution to the DNS security, Fan et al. [8] proposed preventions embedded in security proxies. But their deployment costs are fairly high because security proxies need to be deployed at both authoritative servers and recursive resolvers to support packing and unpacking of all DNS packets with security label. Schomp et al. [9] proposed to remove shared DNS resolvers entirely and leave recursive resolution to the clients. That radical change fails to account for the complexity of DNS clients, the intranet attacks, and the overwhelming pressure on the DNS service providers. Sun et al. [10] proposed DepenDNS as a countermeasure which query multiple resolvers concurrently to verify a trustworthy answer. The reliability and availability of history response data used by DepenDNS is a great concern. Besides, the performance concern about DepenDNS is when the queries are multiplied, their processing overheads will also be multiplied. An extension to DNSSEC was proposed in [15], making

the trust islands verifiable through extended chain of trust. Nevertheless, the overheads of DNSSEC are not lessened by the extension.

Shulman and Waidner [11] performed a critical study of the prominent defense mechanisms against poisoning attacks by off-path adversaries, concluding that existing easy-to-deploy defenses are not so reliable and thus transition to DNSSEC deserves the efforts. The capability of the DNS cache poisoning attacks was studied in [12,13], which are helpful to better understand our proposed defense.

3 The ODD Defense

To "condense" DNSSEC as best as possible while retaining its security capability against cache poisoning attacks, we propose that DNSSEC can coalesce with attack detection to lower its overheads.

3.1 Attack Detection

Off-path cache poisoning attacks are characterized by massive guessing attempts. Cache poisoning is where the attacker manages to inject bogus data into a recursive resolver's cache with carefully crafted and timed DNS packets. A cache poisoned resolver will response with its wrongfully accepted and cached data, redirecting its clients to the bogus and possibly malicious sites. For the sake of being accepted by the target resolver, bogus responses have to guess the transaction ID, port number, and source address of their genuine counterparts.

For one DNS question, an unmatched response satisfies:

(a) It matches the DNS question (or precisely the triple $< qname, qtype, qclass >$) of the outstanding queri(es). Note that attackers may exploit multiple outstanding queries for the same question to significantly increase the success rate of caching poisoning. This is referred to as "birthday attack". In that case, more than one outstanding queries may share one question.
(b) If (a) holds, it mismatches at least one item among transaction ID, port number, and source address of the outstanding queri(es).

A number of unmatched responses with wrong guessing are expected to be found by the target resolver before one bogus response may accidentally succeed. So we propose that presence and accumulating of unmatched responses can be treated as indicator of possible cache poisoning attacks. As a means of attack detection, the recursive resolver counts the incoming unmatched responses for each outstanding DNS question. When the count amounts to a threshold of defense (ToD), the attack traffic is identified and the attack response is triggered.

The appropriate setting of ToD should consider: on one hand, a too large value will result in a non-negligible increase of cache poisoning success rate ahead of any defense in place. e.g., the number of forgery responses is in the order of ten thousands to ensure a 50% chance of compromise in most cases of DNS

```
1: BogusCount ← 0;
2: SEND THE REQUEST;
3: while BogusCount < ToD do
4:    if time out then
5:       RETURN(TIME_OUT);        % The name resolution times out
6:    LISTEN TO THE RESPONSE;
7:    if the response is bogus then
8:       BogusCount ← BogusCount + 1;
9:    else
10:      RETURN(THE RESPONSE);    % The authentic response received
11: SEND THE DNSSEC REQUEST;
12: while not time out do
13:    LISTEN TO THE RESPONSE;
14:    if the response is validated then
15:       RETURN(THE RESPONSE);  % The validating response received
16: RETURN(TIME_OUT);            % The name resolution times out
```

Fig. 1. The responding process of the DNSSEC-aware mode.

operations [12,13]; on the other hand, a too small value will too readily trigger the defense. Problem of false positive stands here when non-malicious or negligent users may unintentionally create a small amount of malformed responses which are identified as unmatched responses. Another exploit of a small threshold is that adversaries may deliberately feed a few unmatched responses on the target resolver in a bid to overload it with excessive defenses.

3.2 DNSSEC-Oblivious Mode

The DNSSEC-oblivious mode lets recursive resolver refrained from sending out DNSSEC-enabled requests nor validating responses unless explicitly required by the client (which sets the DO bit). More than the simple DNSSEC-oblivious DNS, a resolver in the DNSSEC-oblivious mode should perform attack detection and switch to the DNSSEC-aware mode once caching poisoning attack is detected. Therefore the costs of the DNSSEC-oblivious mode are comparable to the simple DNSSEC-oblivious DNS. As long as no caching poisoning attack is detected, the DNSSEC-oblivious mode continues as a normalcy.

3.3 DNSSEC-Aware Mode

The DNSSEC-aware mode uses DNSSEC transactions to authenticate suspicious responses to any potentially targeted DNS question. The responding process of DNSSEC-aware mode is illustrated in Fig. 1. When the count of unmatched bogus responses reaches ToD, the recursive resolver should immediately initiate a separate DNSSEC request for that targeted DNS question. If validated, the response, which is called "validating response" hereinafter, is taken as the trustworthy authority for that question. Thus all valid responses arriving prior to

the validating response are hold on rather than accepted. Note that the hold-on responses may include the genuine response and one or more bogus responses which look like genuine because they totally matches the outstanding question.

3.4 Integration of the Two Modes

We present in detail how the two modes are integrated to defend against cache poisoning attacks. In particular, our example in Fig. 2 shows the defense procedure under the most mighty version of Kaminsky class attacks:

① The attacker client sends the target resolver a query for the IP address of "asq50pn.foo.com" below the target domain "foo.com". The domain "asq50pn.foo.com" is delicately crafted with random characters so that it is likely to miss the resolver's cache to trigger an outstanding query.

②a The forgery authoritative server tries to send cache poisoning attempts to the target resolver guessing the transaction ID, etc. of the genius response. Each unmatched response may, e.g., guess a wrong transaction ID, and intends to inject the IP address of the forgery authoritative server, say "Y.Y.Y.Y".

②b Roughly in parallel with (2a), the target resolver in the DNSSEC-oblivious mode sends a request to the real authoritative name server for "asq50pn.foo.com".

③a When the attack detection count the unmatched responses to ToD, the target resolver switches to the DNSSEC-aware mode and sends a DNSSEC request for "asq50pn.foo.com".

③b Perhaps at the same time as (3a), the genuine response arrives at the target resolver informing the IP address of the real authoritative server, say "X.X.X.X". However, as the DNSSEC-aware mode is already turned on, the response is hold on rather than simply accepted.

③c The target resolver may still persistently be fed with cache poisoning responses in the DNSSEC-aware mode. And the continuous response guessing efforts do have a chance of being holding on.

④ When the validating response is obtained by the target resolver, the relevant records in the validating response are subject to DNSSEC validation using the verified public key. That DNSSEC validation may render further DNSSEC transactions such as step (5) and (6) because some signatures (RRSIG records) over the interested data may be absent from the original validating response.

⑤ The target resolver initiates a new DNSSEC transaction to validate the IP address of the authoritative server ("ns.foo.com").

⑥ The new validating response contains a RRSIG record over the A type (IP address) record of "ns.foo.com". By then, the validating response can be validated.

⑦ By checking the hold-on list against the validating response, the IP address of "ns.foo.com", namely "X.X.X.X", is identified as genuine and "Y.Y.Y.Y" as bogus. The validated record can thus be used by the target resolver in the final answer as well as in the cache.

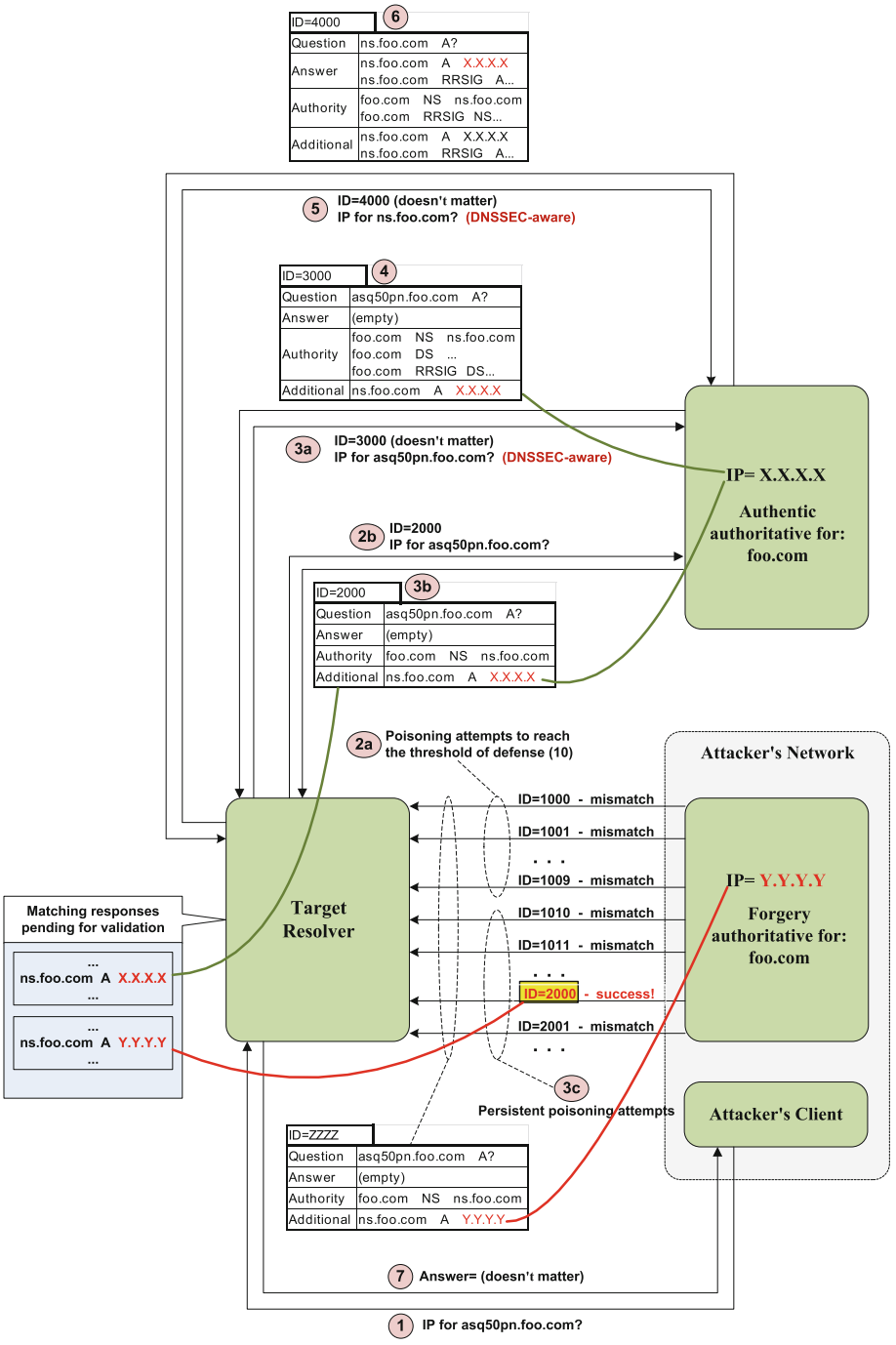

Fig. 2. An example of the integration of the two modes.

3.5 Caching and Proactive Updating of Validating Response

(a) Caching of Validating Response

To overcome the short-lived protection, we propose that recursive resolver should retain validating responses in cache for a long-lived defense rather than just use them once.

The signed records contained in the validating responses and validated by the recursive resolver should be regarded as more trustworthy than the unsigned records in the valid but unsigned responses. Similar to conventional DNS caching, those validating records should be cached by the recursive resolver for a period of TTL (Time-To-Live). Hence the recursive resolver can first search its cache for any relevant validating records before it has to solicit the authoritative servers. Nevertheless, the caching of validating responses differs from conventional DNS caching in the following:

- The validating records are given a priority over the unsigned records, and thus they are stored in a priority cache other than a normal cache. Here "priority" means: a record in the priority cache can overwrite its unsigned counterpart in the normal cache if any conflict exists between them; in turn, a record in the priority cache cannot be overwritten by any unsigned record in a more recent unsigned response; any record in the priority cache can only be replaced by a more recent validating response.
- The records in the priority cache are basically used for validating unsigned responses. When an unsigned response arrives with any record conflicting with the priority cache, the recursive resolver should not accept the response. Instead it waits for its possible successor consistent with the priority cache until timeout.

(b) Proactive Updating of Validating Response

The problem of cache consistency arises if simply respecting the priority of validating records in cache. Consider a more recent unsigned response containing up-to-date records R_u, and the virtually outdated validating records R_v in cache, which conflict with R_u, would deny R_u because R_v are more trustworthy.

For the sake of maintaining strong priority cache consistency, the recursive resolver should seek to proactive update validating response in case of cache inconsistency. The hold-on mechanism specified in DNSSEC-aware mode is slightly changed for caching of validating response. That is, the responses inconsistent with the priority cache are temporally hold on rather than discarded. Because the inconsistent responses may include the genuine response due to cache inconsistency, they are reserved for further validation. To still obtain up-to-date validating records in cache when timeout (indicating the possibility of cache inconsistency), the resolver should acquire a fresh validating response. The new validating response will have two usages: validating the hold-on responses and then returning the validated response if any; updating the corresponding validating records in cache. The responding process for aggressive use of validating response is detailed in Fig. 3.

```
 1: BogusCount ← 0;
 2: SEND THE REQUEST;
 3: while BogusCount < ToD do
 4:    if time out then
 5:       RETURN(TIME_OUT);        % The name resolution times out
 6:    LISTEN TO THE RESPONSE;
 7:    if the response is bogus then
 8:       BogusCount ← BogusCount + 1;
 9:    else if the response is NOT consistent with ValidCache then
10:       break;        % Update the possibly outdated valid cache
11:    else
12:       RETURN(THE RESPONSE);        % The authentic response received
13: SEND THE DNSSEC REQUEST;
14: while not time out do
15:    LISTEN TO THE RESPONSE;
16:    if the response is validated then
17:       ValidResponse ← the response;
18:       USE ValidResponse TO UPDATE ValidCache;
19:       RETURN(THE RESPONSE);        % The validating response received
20: RETURN(TIME_OUT);        % The name resolution times out
```

Fig. 3. The responding process of the DNSSEC-aware mode with caching and proactive updating of validating response.

4 Performance Analysis

4.1 Overheads of DNSSEC Transactions

ODD never initiates DNSSEC transactions unless possible cache poisoning attack is detected at the target resolver. Thus for a vast majority of recursive resolvers which are not constantly targeted by cache poisoning adversaries, ODD is lightweight in terms of name resolution cost at both recursive resolvers and authoritative servers in comparison with the existing DNSSEC deployment strategy.

Consider the worst case of cache poisoning attack. That is, the attacker continuously sends caching poisoning responses at a high rate towards the target resolver. A DNSSEC transaction is generated by the target resolver if and only if:

- The validated records expire from cache so that an immediate flurry of caching poisoning responses triggers the switch to DNSSEC-aware mode.
- No validated response is found until timeout because of the updated authoritative record.

To investigate the event of DNSSEC transactions, we first discuss the events of TTL expiration and the events of authoritative record updating separately. Without loss of generality, we assume the TTL of any validated record follows

a probability distribution function. If the target record is heavily requested, the times between successive events (queries) can be approximated by the value of TTL at the instances of events. Let the TTLs or the successive inter-event times are independently and identically distributed. Then we have

Assumption 1. *There is a renewal process in operation for TTL-expiration-triggered DNSSEC transactions.*

Assume that the successive times between the updates of authoritative records are independently and identically distributed. Then we have

Assumption 2. *There is a renewal process in operation for authoritative-update-triggered DNSSEC transactions.*

The process of DNSSEC transactions initiated by ODD is obtained by superposing the two renewal processes assumed above. However, we can prove the following theorem.

Theorem 1. *The two renewal processes are NOT independent of each other.*

Proof: No matter how long the validating record's TTL elapses, every authoritative-update-triggered DNSSEC transaction should be initiated immediately after the instance of authoritative update (given the intense enough cache poisoning attempts). So the renewal process of authoritative-update-triggered DNSSEC transactions is independent of that of TTL-expiration-triggered DNSSEC transactions. Nevertheless, the renewal process of TTL-expiration-triggered DNSSEC transactions is dependent of that of authoritative-update-triggered DNSSEC transactions. For example, if there is no authoritative update between two successive TTL-expiration-triggered DNSSEC transactions, the inter-even time between the two DNSSEC transactions is roughly TTL; but if there is one authoritative update between them, the residual TTL is renewed to a full TTL at the instance of authoritative update, and so their inter-even time is prolonged to be a full TTL plus a residual TTL; further, if there is more than one authoritative update between them, the residual TTL is renewed more than one time and their inter-even time becomes a full TTL plus more than one residual TTL. □

Given Theorem 1, the process of DNSSEC transactions initiated by ODD cannot be considered to be formed by superposing the two individual renewal processes. Instead, we describe the process of DNSSEC transactions using the codes in Fig. 4.

4.2 Cache Poisoning Success Rate

In Kaminsky cache poisoning attacks, an attacker can balance between the number of outstanding requests and the number of bogus response attempts at will to achieve maximum efficiency [12]. Because the number of effective bogus response attempts is limited by ODD, the attacker often exploits duplicate requests in a bid to increase the probability of successful compromise. However, the number of outstanding requests are bounded by two aspects in practice:

```
1:  % The present time is initialized at an instance of update-triggered query
2:  t ← 0;          % The time is initialized as zero
3:  T ← TTL;        % The residual TTL is a TTL after an update-triggered query
4:  while True do
5:    if T = 0 then
6:      SEND A REQUEST;          % Initiate a TTL-triggered query
7:      T ← TTL;
8:    else if an authoritative update occurs at t then
9:      SEND A REQUEST;          % Initiate an update-triggered query
10:     T ← TTL;
11:   t ← ELAPSE(t);             % Time elapses
12:   T ← T−(ELAPSE(T)−T);   % The residual TTL decreases as time elapses
```

Fig. 4. The process of DNSSEC query event by ODD.

- The maximum number of outstanding requests is set as a default configuration in some widely used authoritative server implementations. Authoritative servers thereby discard excessive outstanding requests surpassing the configured limit, say L_a. So any efforts of producing more than L_a outstanding requests will prove fruitless [13].
- The window allowed to persistently elicit outstanding requests is bounded by the response time T_r perceived by the target resolver. Let the average query sending rate of attacker be R. The window can be converted to the number of outstanding requests roughly as T_r/R. In summary, the maximum number of outstanding requests D is the minimum of the two limits, namely $D = min\{L_a, T_r/R\}$.

Within one round of ODD validation, there are at most ToD-1 bogus response attempts left for effective cache poisoning. Letting H =ToD-1, we can express the cumulative probability of cache poisoning failure in all attempts up to and including the H th attempt as

$$P_D(H) = P(\textit{the } 1\textit{st attempt misses, the } 2\textit{nd attempt misses, ..., the } H \textit{ th attempt misses} \mid D \textit{ identical outstanding queries}) \tag{1}$$

Suppose the number distinct IDs available I, the number of ports used P, and the number of authoritative servers for a domain N. If $H \ll (I+P)*N$, $P_D(H)$ can be written as

$$P_D(H) = (1 - D/((I+P)*N))^H \tag{2}$$

The worst case of ODD validation is when no relevant validating record is available at cache and thus a DNSSEC transaction is initiated for it. And then the validating record fetched is cached for it TTL to protect from any further cache poisoning attempts. Being the minimum window of opportunity for H attempts, the interval can be approximated by two response times, one for the

proceeding non-DNSSEC response and the other for the following validating response, plus the TTL of validating record. So we have

$$T_H = 2 * T_r + TTL \tag{3}$$

where T_H denotes the minimum window of opportunity for H attempts and TTL denotes the TTL of validating record. That is, one round of ODD validation takes at least two response times plus one TTL to obtain a success rate of $1 - P_D(H)$. The success rate of cache poisoning within i rounds of cache poisoning attempt is $1 - P_D(H)^i$.

As illustrated in Fig. 4, the duration of defense by validating record in cache may be further prolonged to more than TTL. That extension to the window of opportunity occurs if authoritative update is identified by the resolver to refresh the validating record in cache before its TTL expires. In such case, the continuous elapse of TTL is interrupted by any authoritative update which renews the residual TTL to a full TTL. Therefore the effects of window extension are better pronounced for more frequent authoritative update, which provides a better chance of repeated TTL renewals.

Table 1. Parameters and their settings.

Parameter	Setting
Number distinct IDs available (I)	65536
Number of ports used (less than 1024 are unavailable) (P)	64000
Number of authoritative servers for a domain (N)	2.5
Response time (T_r)	0.02 s
Number of identical outstanding queries (D)	20
Query sending rate from resolver to authoritative server	100 qps
Query responding rate from authoritative server to resolver	100 qps
Query sending rate from attacker to resolver (R)	1000 qps
ToD	3
Bogus responding rate from attacker to resolver	100
Minimum window of opportunity for H attempts (T_H)	10 h

5 Model Checking Results

Probabilistic model checking is one of the most commonly used formal verification technique for the modeling and analysis of stochastic systems. We model Kaminsky cache poisoning attack as a continuous-time Markov chain (CTMC) using PRISM [14]. In modeling the attack, we assume that the queries originated from the attacker look up a random generated domain such that they will never hit the target resolver's cache. We also assume that the IP addresses of the target domain's authoritative servers are always in the cache of the target resolver.

5.1 Results of Query Load

To investigate the combined effects of TTL expiration and authoritative update on the inter-time of DNSSEC queries, we generate a sequence of authoritative update events following a probabilistic distribution while setting the TTLs in the DNSSEC responses as constant and probabilistic values respectively. The inter-time of authoritative updates follows exponential distribution. We use Monte Carlo method to estimate the mean of inter-times of DNSSEC queries. In each experiment, 100,000 times of authoritative updates are generated from an exponential distribution. A number of TTLs, taking either constant values or probabilistic values, are also produced to cover the same time span at the instances when the predecessor TTL expires or authoritative update takes place.

Figure 5 illustrates how DNSSEC query intervals change with authoritative update intervals. We can see that a very small authoritative update interval has almost the same DNSSEC query interval because TTL expiration rarely happens. But for a larger authoritative update interval, the effect of TTL expiration is better pronounced because a TTL has more chance of being smaller than an authoritative update interval thus more chance of expiration. Random TTLs, though have the same mean as constant TTLs, tend to cause a slightly larger DNSSEC query intervals and thereby a smaller DNSSEC query load on authoritative servers. The ratio of TTL-expiration-triggered queries is illustrated in Fig. 6. We can see that the ratio of TTL-expiration-triggered queries grows as the mean of update intervals increases. But authoritative update tends to pronounce more than TTL expiration on triggering DNSSEC queries even if they share the same mean interval. As shown in Fig. 7, when both update interval and TTL take a mean of 1000s, TTL-expiration-triggered DNSSEC queries only account for about 36% of the total. That can be explained by the fact that the event of authoritative update is independent of and never superseded by the event of TTL expiration while the arrival of TTL expiration may be interrupted and renewed by authoritative updates.

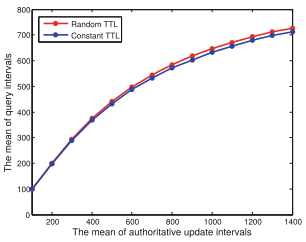

Fig. 5. DNSSEC query intervals vs authoritative update intervals.

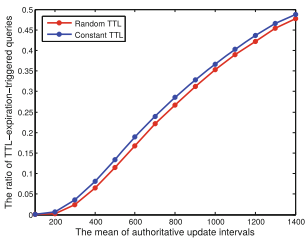

Fig. 6. Ratio of TTL-triggered queries vs authoritative update intervals.

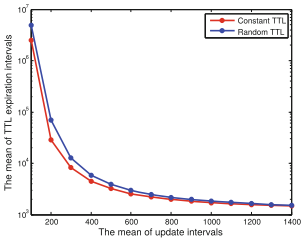

Fig. 7. TTL expiration intervals vs authoritative update intervals.

It is obvious that DNSSEC query interval will be larger if authoritative update and TTL expiration are independent. So in order to examine the lower bound of DNSSEC query interval or the upper bound of DNSSEC query rate, we assume that authoritative update and TTL expiration are independent. Then the mean DNSSEC query interval can be written as

$$I_{overall} = \frac{I_{update} * I_{ttl}}{I_{update} + I_{ttl}} \tag{4}$$

where I_{update} and I_{ttl} represent authoritative update interval and TTL respectively. As illustrated in Figs. 5 and 6, we conclude that the maximum DNSSEC query rate of ODD under intense cache poisoning attempts is of the same order as the minimum of authoritative update rate and the reciprocal of TTL.

5.2 Results of Cache Poisoning Success Rate

We configure the default values in Table 1 for the parameters in the model checking unless their values are otherwise defined.

First, we illustrate the time needed for a 50% success rate under different minimum window of opportunity in Fig. 8 (ToD=3). We can see that the time cost of cache poisoning roughly grows linearly with minimum window of opportunity. For a minimum window of opportunity above 10 h, the time required for a 50% success rate amounts to no less than 2 years. This is because the longer are the validating records available in cache to defend against cache poisoning attacks, the longer does an attacker have to wait to embark the next round of cache poisoning attempts (if the current round fails). As the TTLs of many authoritative records are set in the order of days or even weeks, it is very hard in practice to compromise them through cache poisoning attacks. Figure 8 also shows creating more identical outstanding queries may dramatically decrease the difficulty of cache poisoning. Thus in the defense, the resolver should not allow excessive identical outstanding queries in order to prevent an unacceptable success rate of cache poisoning.

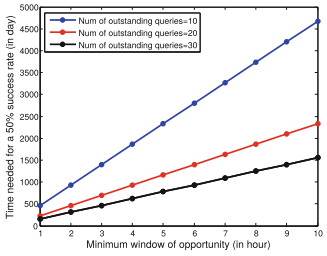

Fig. 8. Time needed for a 50% success rate vs minimum window of opportunity (ToD = 3).

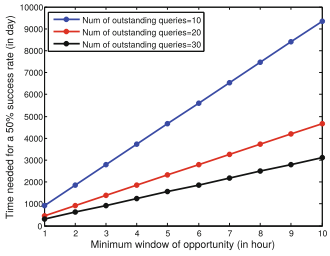

Fig. 9. Time needed for a 50% success rate vs minimum window of opportunity (ToD = 2).

Second, we investigate the impacts of ToD on the success rate. In Fig. 9, the time needed for a 50% success rate is shown when the ToD is lowered to 2. We can see that limiting ToD helps significantly to suppress the success rate of cache poisoning. Since ToD defines the maximum number of forgery responses (ToD-1) allowed without defense, a larger ToD means more chance of guessing attempts thus a larger success rate. To ensure the efficacy of ODD, ToD should be set as a sound small value.

Third, we study how the cache poisoning success rate evolves over time. In Fig. 10, we can see that the success rate over time grows like a stair-step shape. In the curve, each step virtually represents a cache poisoning attempt in time and an accumulation of ToD-1 forgery responses in success rate. And the width of each stair-step is dominated by minimum window of opportunity. When ToD is three in Fig. 10, there are two forgery responses aggregated in a round of cache poisoning attempts to increase the overall success rate.

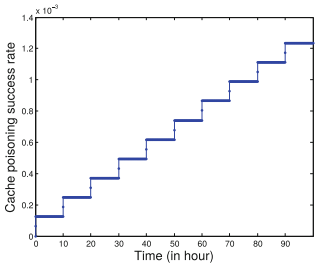

Fig. 10. Cache poisoning success rate vs time (ToD = 3).

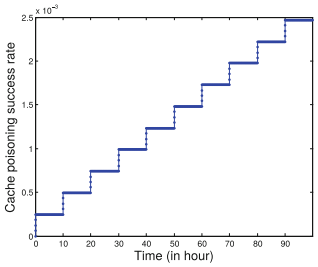

Fig. 11. Cache poisoning success rate vs time (ToD = 5).

Fourth, how the setting of ToD impacts the cache poisoning success rate is studied. As illustrated in Fig. 11, the increase of ToD from 3 to 5 will lessen the defense of ODD against cache poisoning attacks. While the width of each stair-step stays the same as Fig. 10, the jump of each stair-step in the success rate is doubled. So the overall success rate grows much faster than Fig. 10. This shows again that a large ToD may undermine the defense capability of ODD.

6 Conclusions

DNSSEC deployment suffers from its significant costs which slow its progress. To speed up DNSSEC adoption, a lightweight DNSSEC solution was proposed. The proposed ODD defense greatly lowers the DNSSEC overheads while reserving the DNSSEC defense capability against cache poisoning attacks. Because of its efficiency and efficacy, ODD can serve as an interim mechanism for speeding DNSSEC adoption over a long-term transition to DNSSEC.

References

1. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: Resource records for the DNS security extensions. In: RFC 4034, March 2005
2. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: Protocol modifications for the DNS security extensions. In: RFC 4035, March 2005
3. Kaminsky, D.: It's the end of the cache as we know it. In: BlackHat (2008)
4. Huston, G., Michaelson, G.: Measuring DNSSEC performance (2013). http://www.potaroo.net/ispcol/2013-05/dnssec-performance.pdf
5. Migault, D., Girard, C., Laurent, M.: A performance view on DNSSEC migration. In: Proceedings of the International Conference on Network and Service Management (CNSM 2010), pp. 469–474 (2010)
6. Ager, B., Dreger, H., Feldmann, A.: Predicting the DNSSEC overhead using DNS traces. In: Proceedings of the Conference on Information Sciences and Systems (CISS 2006), pp. 1484–1489 (2006)
7. Lian, W., Rescorla, E., Shacham, H., Savage, S.: Measuring the practical impact of DNSSEC deployment. In: Proceedings of the USENIX SEC 2013, pp. 573–588 (2013)
8. Fan, L., Wang, Y., Cheng, X., Li, J.: Prevent DNS cache poisoning using security proxy. In: Proceedings of theInternational Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2011), pp. 387–393 (2011)
9. Schomp, K., Allman, M., Rabinovich, M.: DNS resolvers considered harmful. In: Proceedings of the ACM HotNets 2014, pp. 16–22 (2014)
10. Sun, H.-M., Chang, W.-H., Chang, S.-Y., Lin, Y.-H.: DepenDNS: dependable mechanism against DNS cache poisoning. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 174–188. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10433-6_12
11. Shulman, H., Waidner, M.: Towards forensic analysis of attacks with DNSSEC. In: Proceedings of the IEEE Security and Privacy Workshops (SPW 2014), pp. 69–76 (2014)
12. Wang, Z.: POSTER: on the capability of DNS cache poisoning attacks. In: Proceedings of the ACM CCS 2014, pp. 1523–1525 (2014)
13. Wang, Z.: A revisit of DNS Kaminsky cache poisoning attacks. In: Proceedings of the IEEE GLOBECOM 2015, pp. 1–6 (2015)
14. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
15. Wang, Z., Rose, S., Huang, J.: Securing DNS-based CDN request routing. IEEE COMSOC MMTC Commun. - Front. **12**(2), 45–49 (2017)