Recursive DNS Architectures and Vulnerability Implications

David Dagon[1], Manos Antonakakis[1], Kevin Day[2], Xiapu Luo[1], Christopher P. Lee[3], and Wenke Lee[1]

[1]College of Computing, Georgia Institute of Technology,
{dagon,manos,csxpluo,wenke}@cc.gatech.edu

[2] your.org
kevin@your.org

[3]College of Engineering, Georgia Institute of Technology,
chrislee@gatech.edu

Abstract

DNS implementers face numerous choices in architecting DNS resolvers, each with profound implications for security. Absent the use of DNSSEC, there are numerous interim techniques to improve DNS forgery resistance. We explore how different resolver architectures can affect the risk of DNS poisoning.

The contributions of this work include: (A) We create a comprehensive, accurate model of DNS poisoning. We show how this model is more sensitive than other previous explanations of DNS poisoning. (B) We further catalog the major architectural choices DNS implementers can make in query management. We note real-world instances where these choices have weakened the security of resolvers, and measure the impact on security using our model. Our study revealed numerous, previously unknown vulnerabilities in common DNS servers.

1 Introduction

The Domain Name System (DNS) [32, 33] plays a critical and often unexamined role in Internet communications. The importance of DNS comes from two essential properties:

- *DNS resolution happens first*, as a precursor to almost every other protocol, and
- with few exceptions, successful DNS resolution is essential to the operation of all other protocols (mail, web, VOIP). Without a working, secure resolution system, other protocols are at risk.

Security problems have plagued DNS implementations for years. One class of security flaws is DNS poisoning, where remote attackers forge DNS responses to pollute recursive resolvers with malicious records. DNS poisoning attacks can transcend vendor-specific implementations. Most recently, security researcher Dan Kaminsky [23] noted a flaw in the DNS protocol that allowed quick, deterministic DNS poisoning of nearly every RFC-compliant resolver. This required the rapid update of millions of DNS servers, used by hundreds of millions of users—a near replacement of the existing DNS footprint on the Internet.

In a general sense, there are two classes of complementary solutions to DNS poisoning: long-term improvements to resist forgery attacks that implement cryptographic protections and short-term solutions that help the Internet transition to more secure architectures. Long term solutions, such technologies as DNSSEC [6–8] and DNSCurve [13], generally require replacement of *both* the recursive and authority servers of the existing DNS tree. This has proven to be a complex, multi-year process. By contrast, solutions that affect just the recursive servers have proven easier to adopt. For example, source port randomization (to mitigate the Kaminsky-class of poisoning attacks) only requires a change to recursive server software.

Despite efforts to correct known bugs, there is a high potential for further vulnerabilities in the DNS protocol and specific DNS server implementations. This potential has driven interest in interim security solutions—technologies that fall short of DNSSEC, but still make servers more resistant to DNS forgeries. DNS-0x20, for example, is an encoding technique used by recursive domain name servers and makes it harder, but not impossible, to poison caches [15]. By randomly flipping upper and lower case characters in domain names (e.g. `example.com` transformed to `ExAMpLe.CoM`), and observing the identical pattern repeated in the answer, this technique introduces an extra source of entropy for the DNS transactions. For the most part, DNS-0x20 requires only changes to the DNS initiator's recursive; the stub resolver is unaware of the encoding scheme. Having DNS-0x20 functionality in recursives significantly improves the entropy for all iterative DNS transactions. One drawback is that the entropy increases exponentially to the length of the domain name. Consequently, small domain names (e.g `ibm.com`) and domain names with few 0x20 capable characters (e.g., `163.com`) will not significantly benefit from DNS-0x20.

The IETF working group on DNS, DNS Extensions (DNS-EXT) [19], has seen a tremendous number of proposals that improve DNS security and make recursive resolution more resistant to poisoning. The proliferation of DNS protective solutions creates numerous options for DNS implementers: they can choose between DNS-0x20, source port randomization (SPR), tighter bailiwick logic [10], or other interim fixes. These solutions are described in other papers and IETF proposals [2, 15]. In general, because some solutions result in unwanted "on path" resolution failures, or come with high resource costs, not every vendor will pick the same mix of interim technologies. Source port randomization, for example, has proven to be resource intensive and may not be appropriate for marginal platforms (e.g., embedded devices offering DNS resolution). DNS-0x20 works in nearly every case, but some authority servers employ non-compliant server load balancers, and some domains realize small benefits. These *costs* can be measured, but without a way to measure security *improvements*, the task of selecting appropriate DNS forgery resistance technologies for a given platform becomes extremely difficult.

With the exception of bailiwick checking logic (which creates simple boolean conditions of cache/don't-cache), this paper provides a model of all of these security improvements to DNS. Specifically, this paper provides the following contributions: ***First***, we discuss in detail recent protocol vulnerabilities in DNS and note how these motivate the need for interim DNS security technologies. ***Second***, we provide a comprehensive model of DNS poisoning, so that on-path data attacks in DNS can be fully understood, outside of any specific vendor or technological context. ***Third***, we survey the various interim DNS security technologies and use our model to describe the security benefits to DNS architects. To demonstrate the utility of our approach, our survey discovered several previously unknown vulnerabilities in DNS resolvers. We further worked with vendors to provide patches.

Section 2 provides a brief background on DNS resolution and vulnerability vectors. Section 3 discusses our DNS poisoning model in detail and compares it to other previous models. In Section 4, we conduct a survey of DNS systems and discover numerous instances of DNS poisonings along with a large pool of DNS servers still vulnerable to trivial attack. Our findings show the need to more short-term interim security improvements to DNS, ahead of DNSSEC. We use our model of DNS poisoning to analyze options for improving selected DNS resolvers. The appendix provides a script-based implementation of our model, so that DNS developers can evaluate the benefits of adopting specific interim security fixes.

2 Background

We provide a brief overview of the Domain Name System (DNS), an infrastructure commonly used to map domain names to IP addresses. We focus only on those aspects of DNS relevant to DNS poisoning and defensive architectures. A general and readable overview of DNS is found in [43].

DNS uses a tree structure to organize domain name-space into a distributed database. A domain is a node in this tree, with each label separated by a period.

A *zone* is a clique of nodes. The clique of nodes form a contiguous tree structure, the top of which is called the *start of authority*. Authority DNS servers answer queries about their zones, either providing the mappings for *leaf nodes* or answering with referrals that indicate delegation of child zones to other authority servers.

For our purposes, there are four fields in any DNS message—both queries and answers. The fields are the query, the answer (which include RRset mappings between domains and IPs), the authority field, and the additional field. The authority field is used in the case of referral answers. For example, an authority for a zone can indicate downward delegation of authority. With many exceptions (e.g., in the case of an `NS` query), the authority field is used to provide mappings between authorities for zones and hosts. This takes the form of "glue records," IP mappings of nameservers for child zones. Ultimately, when an authority server answers directly from its zone, it signals that the answer is "authoritative" by setting header flags.

To protect against malicious insertion of untrustworthy authority records, name servers typically require answers to be "in bailiwick." That is, they insist that the authority record be in the same zone cut as the query. The reasoning is that a server should not be trusted to provide answers about sibling zones—only child labels. Thus, if one queries for `host.example.com`, answers that provide authority records for out-of-bailiwick zones (such as `www.google.com`), are not trusted. Instead, recursive resolvers will iteratively re-query for the desired nameserver.

RFC 2181 [18] describes a range of trustworthiness for data found in DNS answers. The most trustworthy information in a DNS message (other than direct access to a zone file or an AXFR) comes from authority data from an authority server. The least trustworthy data, according to RFC 2181, comes from additional information provided by authority servers. Critically, RFC 2181 reaffirmed a reference in RFC 1034 4.3.4 (concerning negative caching) that the "authority section of an authoritative answer may contain the SOA record for the zone

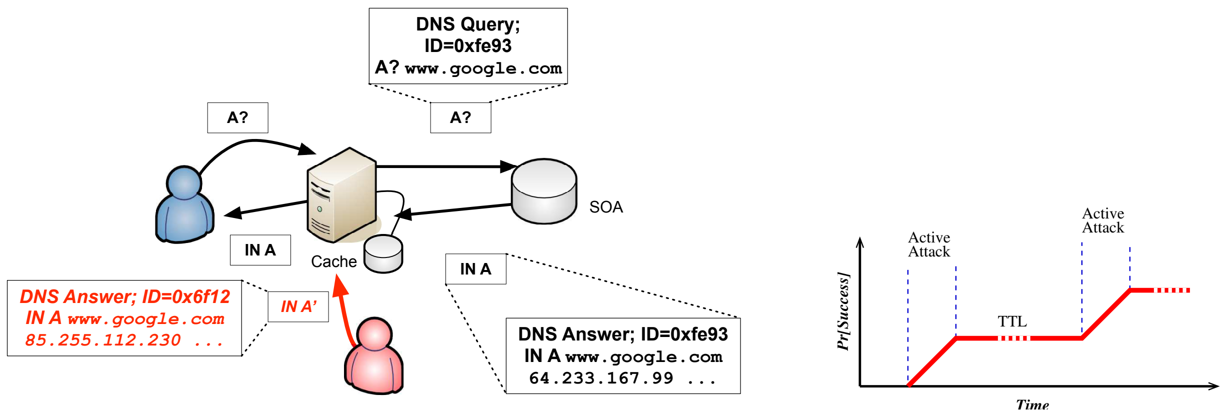


Figure 1. (a) Simplified DNS poisoning attack. (b) Cumulative probability over time for an attacker to poison a DNS cache.

from which the answer was obtained." This property later became the basis for the Kaminsky-class poisoning attack, which is discussed below.

To help illustrate this process, Figure 1(a) shows selected portions of a typical DNS query. A user begins with a query (abbreviated as `A? <domain>` in DNS nomenclature) from a stub resolver to a recursive resolver. The recursive resolver surfs the zone hierarchy to locate the appropriate authority for the desired zone containing the domain and then iteratively obtains the answer (noted as `IN A`).

2.1 DNS Poisoning Attacks

Figure 1(a) also illustrates typical attacks on DNS resolution. Of recent concern is the class of DNS attacks generally known as DNS poisoning. These attacks, discussed in detail in [15], generally follow a pattern:

- an attacker forces a recursive resolver to initiate a DNS query,
- while the recursive resolver is waiting for the authority resolver's answer, the attacker forges numerous answers to guess the transaction elements used in the recursive resolver's message. Conceptually, this is a packet race between the correct answer and any successful guess the attacker can send to the recursive.

With only 16-bit ID fields in the base DNS protocol, one would expect to see more instances of DNS poisoning. Figure 1(b) illustrates why, with only 16-bits of entropy, DNS transactions were not previously widely abused. The cumulative probability of a successful poisoning is plotted on the y-axis of Figure 1(b), from 0 to 1. Each spoofed answer has, in the simplest case, a $\frac{1}{65{,}536}$ chance of matching the transaction components of the recursive's query. As more attack packets are sent over time, the attacker's cumulative chance of success improves, eventually reaching 1 when all $65,536$ possible transactions values are guessed. But since DNS poisoning is a type of packet race, often the correct answer will arrive and be cached by the recursive server. The dimension of time, shown on the x-axis of Figure 1(b) is critical. The time between a recursive resolver's query and the response from the authoritative DNS server is about 100 milliseconds on the average (with various local maxima, as noted in [15]). But TTL periods are (with some exceptions) typically days in length. Thus, each time a round of spoofed packets (sent within milliseconds) fails to poison a recursive resolver, the attacker must wait out the TTL period (days) before trying again. Though they may eventually win the packet race, it may take weeks or longer, depending on the cache period of the DNS server, random restarts of the cache, random cache eviction events followed by repeated queries, and other developments.

For this reason, DNS poisoning has historically proved difficult, unless there were implementation errors in the DNS server's random number generator [25–29]. However, researcher Dan Kaminsky recently observed a possible accelerator for simple DNS poisoning. Figure 2(a) illustrates the operation of this attack. Instead of directly poisoning the answer field of a query, the attacker provides a malicious authority field in guessed answers. The attacker is free to query for random child labels of a zone and, when a successful guess is made, the meaningless answer is cached along with the malicious `NS` record.

To poison the NS record for `example.com`, for example, an attacker queries the victim recursive server for `A?` $\$RANDOM_1$`.example.com`, and then spoof answers that contain two parts. The first part has some arbitrary answer to the query and the second part an authority answer that claims a malicious IP is the new `NS` for the zone. If the attack is unsuccessful and the correct answer (likely `NXDOMAIN`) arrives first, the attacker immediately asks the victim recursive for `A?` $\$RANDOM_2$`.example.com`, until successfully guessing the transaction components of the recursive query. The malicious component of the attack messages lies in the authority update, not the answer field. Superficially, this is similar to the Kashpureff poisoning attack of the 1990s, which padded

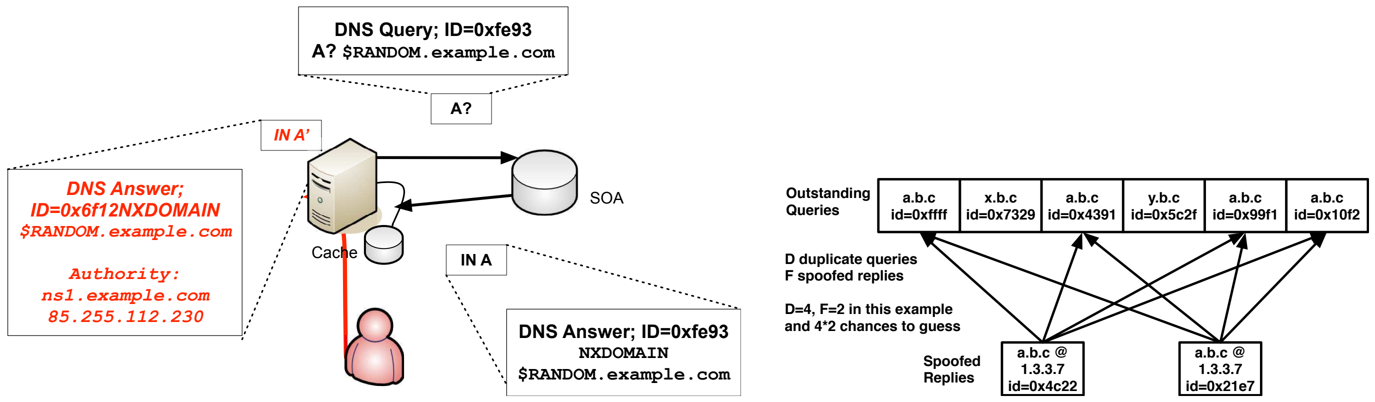


Figure 2. (a) Kaminsky-class poisoning where the attacker repeated uses random label within a domain in attempt to continually spoof answers (b) Conceptual view of the Birthday Attack on DNS Resolution.

malicious "additional record" answers onto spoofed DNS answers [42].

This attack strategy revolutionized DNS poisoning attacks and reduced the time-to-success from weeks to mere seconds. Since an attacker could always become a zone's authority, the consequences of such an attack were potentially more severe than simple A-record manipulation. The attack enabled trivial man-in-the-middle attacks, email manipulation/interception, and VOIP tapping. Since Kaminsky-class poisonings allow attackers to make unlimited guesses, cache manipulation is no longer bounded by TTL parameters. Such attacks are now (effectively) bandwidth limited. In response, a multivendor patch was released [41], which uses the general strategy of growing the key space by using source port randomization (SPR).

The DNS birthday attack [38] (seen in Figure 2(b)) works by convincing the recursive DNS server to make multiple, concurrent queries for the same label and then spoofing replies for that label. This allows the spoofed packets to collide with any one of the valid port and ID combinations to poison the cache. If the recursive performs D outbound queries for the same label, then each spoofed packet from the attacker has D chances to be right. Since this discovery, many DNS recursive server implementations have adopted *birthday protection* to suppress multiple queries for the same domain, with notable exceptions.

As a clarification point we should mention a small variation between the traditional "birthday problem" and the "DNS birthday problem". The traditional "birthday problem" considers the probability of collision among a set uniformly random n numbers. In the "DNS birthday problem" the set of uniformly random numbers n is comprised from the (d, f) combinations of the DNS transactions and their characteristics, where d is the number of simultaneous queries originated from the recursive DNS server (RDNS) and f the simultaneous attack packets targeting the RDNS. For the "DNS birthday problem" we are interested in collisions between distinct values of d and f in all possible pairs of (d_i, f_j), where i, j are in the message space of d and f, respectively. Additionally, the actual time window in the "DNS birthday problem" is the period from the epoch when an RDNS sends the queries (d) to the epoch when it receives the responses from an authoritative DNS. In section 3, we conduct an in depth analysis for various instances of f and d and their combinations. In the same section, we consider what other defenses a host may use to defend against DNS poisoning and provide a model to evaluate the relative strength of each approach. While DNSSEC and related technologies may provide an ultimate solution, we argue that our survey of DNS attacks counsels for an improved understanding of short-term responses. In Section 4, we track the deployment of source port randomization (SPR) as a response to the Kaminsky-class attack. We note that, months after patches became available, a large number of hosts evidently do not exhibit SPR. We further observe numerous instances of DNS poisoning on the Internet and correlate these with the release of the Kaminsky-class NS-replacement technique. This evidence of attacks, combined with a lack of patching, illustrates the importance of interim DNS security measures like DNS-0x20.

3 A Comprehensive DNS Poisoning Model

In this section, we investigate the trade-off between the probability that a victim DNS's cache is poisoned and the effort that an attacker has to spend to launch a successful Kaminsky-class attack (which is currently the worst-case attack). We propose a precise model of poisoning risk and identify key parameters for calculating the risk of poisoning under various defensive scenarios. We also note how our model improves upon general models proposed by others.

3.1 Parameters Considered in the Attack Model

According to existing DNS standards (i.e. RFC 1034 [32] and RFC 1035 [33]), the randomness in a DNS query comes from the following sources. (Most symbols are the same as those used in [2] for the ease of comparison.)

1. I denotes the ID field in a DNS message's header section. This field, often called the `qid`, is a 16 bit identifier and has $2^{16} = 65,536$ possible values and is used to map answers back to the request.

2. P denotes the number of UDP source ports used by the resolver and is often abbreviated in DNS nomenclature as `sport`. Per RFC 768, this field contains 16 bits, but to avoid collision with other applications, not all source ports can be used in practice. Servers implementing Source Port Randomization (SPR) generally follow these schemes:

   (a) Without using the well known ports (i.e. $0 \rightarrow 1023$) defined by IANA [20], there are $2^{16} - 1024 = 64,512$ available ports for a query.

   (b) According to the list of port numbers updated on 2008-08-01 [20], $52,209$ UDP ports have been formally assigned to various services by IANA. Since the range for dynamic and/or private ports defined by IANA is from $49,512 \rightarrow 65,535$, the resolver following this rule can only select a source port from $16,384$ remaining candidates.

   (c) Implementation faults in some operating systems/DNS allow the attacker to easily guess the source port in a query. For example, some DNS software uses fixed source port (e.g. 53). Although some DNS server will use a wide range of source ports, the port used by a query is predictable (e.g., incrementing) or chosen from a small pool of random ports selected at startup. Moreover, the random number generator in some operating systems is not strong enough to produce high-quality random numbers [30].

   (d) Most Kaminsky-class attack programs, e.g., [14, 21], assume recursive servers randomly select a fixed `sport` at start-up (e.g., as done in unpatched BIND-9.4), with a value from the range $2^{16} - 1024 = 64,512$. Since this port is fixed for the attack, this generation of tools are trivially remedied by SPR.

3. N denotes a compound variable: the product of the number of authoritative nameservers and the number of routable external IPs used by the recursive server. Many DNS authority servers use multiple hosts (as recommended, for example, by RFC 1912 [9]). A resolver can randomly select one and send a query to it. Similarly, a recursive server can use any number of source IP addresses, and select one at random to iteratively query authority hosts. While this is less common, it does provide some additional protection against poisoning attacks. As discussed in [15] the variable N is the product of all IP addresses (both authority and recursive) that can be used in a DNS transaction.

The following parameters affect the probability of compromise:

1. $\mathfrak{D}$ denotes the set of *equivalent* outstanding queries sent by the resolver. Equivalent queries mean that they have the same `qname`. Let $\overline{D}$ be the number of queries in $\mathfrak{D}$. It is determined by several factors:

   (a) The number of original queries sent by the attacker.

   (b) The resolver's bandwidth.

   (c) Adjustable parameters in DNS software (e.g. 200 in dnscache [11] or 1,000 in BIND 9 [22])

   (d) The number of 0x20-bit capable characters in a query.

   Due to the weakness in the random number generator used by a DNS server [3], it is possible that two equivalent queries may also have identical `qid`, source port, and destination IP address. In this case, we consider these two queries *the same*. Therefore, we let D denote the number of *distinct* queries in $\mathfrak{D}$. Although these queries have the same `qname`, they are usually different in their `qid` or source port or the destination IP address (if there are more than one authority name servers, or ANS). Thus, $D = \overline{D}$. But we will consider both the case when $D = \overline{D}$ and the case $D < \overline{D}$ in order to obtain a comprehensive model. Please note that the birthday attack becomes possible when $\overline{D} > 1$.

2. W denotes window of opportunity in seconds. This is the period of time during which a recursive server is anticipating a response: the moment right after the resolver sends a query, to the moment right before the authority server's response arrives. In general, as noted in [15], this period of time is approximately 100 milliseconds, on

the average, though longer events (such as SERVFAIL timeouts) can occur. In some cases, the attacker can arrange for the authority to be unreachable or lagged from the recursive server's network.

3. $\mathfrak{F}$ denotes the set of spoofed responses sent by the attacker. Let $\overline{F}$ be the number of responses in $\mathfrak{F}$. In theory, it is possible that two responses have the same qid, destination port number, and source IP address because of the flaws in the attacker's program, such as repeated random numbers. Although an attacker can use many approaches to make sure that such case will not happen, we still define F as the number of distinct responses in $\mathfrak{F}$ in order to derive a comprehensive model.

4. S_{res} denotes the size of forged response packets. This parameter measures the number of bits in spoofed response packets (to later express risk in terms of bandwidth).

5. R denotes the number of packets sent per second by the attacker. This parameter is limited by the attacker's bandwidth. Note that with some attack platforms, such as botnets, this parameter may be effectively unlimited, or bounded only by the recursive server's network-facing transit.

6. $\mathfrak{R}$ denotes the rate of attack traffic in Mb/s (i.e.) $\mathfrak{R} = RS_{res}$.

7. A denotes the number of attack windows. The attacker may continue the attack until the cache is poisoned. A new attack window is opened anytime the attacker has the recursive server make a unique outbound query.

3.2 The Probability of a Successful Attack

Note that the attack will succeed if at least one out of $\overline{F}$ forged responses matches one out of $\overline{D}$ queries within a window of opportunity, W.

Let $M = I * P * N$. That is, M is the number of total possible choices that a recursive server can use for query. We use $P_s(M, \mathfrak{F}, \mathfrak{D})$ and $P_f(M, \mathfrak{F}, \mathfrak{D})$ to denote the success probability and the failure probability of an attack (i.e. $P_s(M, \mathfrak{F}, \mathfrak{D}) = 1 - P_f(M, \mathfrak{F}, \mathfrak{D})$).

According to the relationship between $\overline{D}$ and D and the relationship between $\overline{F}$ and F, the general model consists of four scenarios:

1. $D = \overline{D}$ and $F = \overline{F}$: most cases fall into this scenario where although all queries have the same qname they are distinguishable according to their qid or source port or the destination IP address if $N > 1$. Similarly, all responses are different.

2. $D < \overline{D}$ and $F = \overline{F}$: in this scenario the DNS server may generate at least two identical queries. For example, when a DNS server uses fixed source port and one ANS, a bad random number generator may produce two identical qids.

3. $D = \overline{D}$ and $F < \overline{F}$: in this scenario an attacker may send at least two identical forged responses.

4. $D < \overline{D}$ and $F < \overline{F}$: this is a general scenario where a DNS server may produce identical queries and an attacker may create identical responses.

To better explain the deduction procedure, we will discuss these scenarios one-by-one. Please note that when considering the scenarios of $D = \overline{D}$ or $F = \overline{F}$ we have two approaches: one is selection with replacement and the other is selection without replacement. But when considering the scenarios of $D < \overline{D}$ or $F < \overline{F}$ we can only use the approach of selection with replacement because selection without replacement will not cause identical queries or identical responses. To follow a unified approach, we only examine selection with replacement in the analysis.

According to the pigeonhole principle, $P_f(M, \mathfrak{F}, \mathfrak{D}) = 0$ if $D + F > M$. In other words, the spoofed responses will match at least one query. This conclusion is suitable for all scenarios.

3.2.1 If $D = \overline{D}$ and $F = \overline{F}$

we have

$$P_f(M, \mathfrak{F}, \mathfrak{D}) = \frac{\binom{M-D}{F}}{\binom{M}{F}} \tag{1}$$

The basic idea is that if F forged responses are selected from the $M - D$ possible responses, then a match between the responses and the queries does not exist. According to [44], Eqn(1) could be approximated by

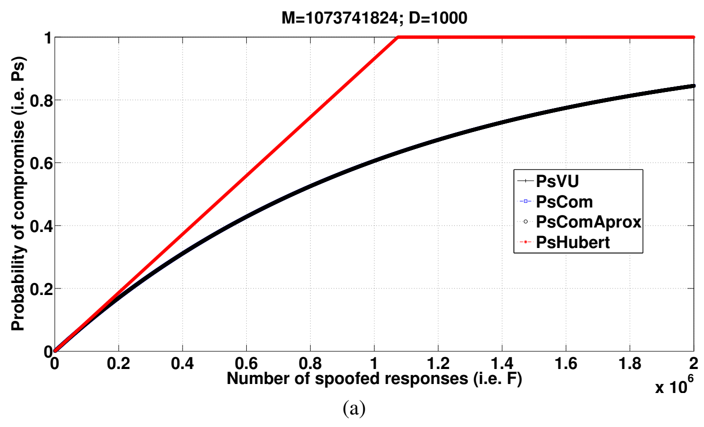
$$P_f(M, \mathfrak{F}, \mathfrak{D}) \approx (1 - \frac{1}{M})^{DF} \tag{2}$$

We will evaluate this approximation in Section 3.3.

3.2.2 If $D < \overline{D}$ and $F = \overline{F}$

We first compute the probability that there are i distinct queries in $\mathfrak{D}$ and denote it as $P(D = i)$, $1 \leq i \leq \overline{D}$. The key point here is to calculate the number of all possible mappings from $\overline{D}$ elements to i elements. We use the method of enumerating subjections [34, 35], and obtain:

$$P(D = i) = i! \mathbb{S}(\overline{D}, i) \frac{\binom{M}{i}}{M^{\overline{D}}}, \tag{3}$$



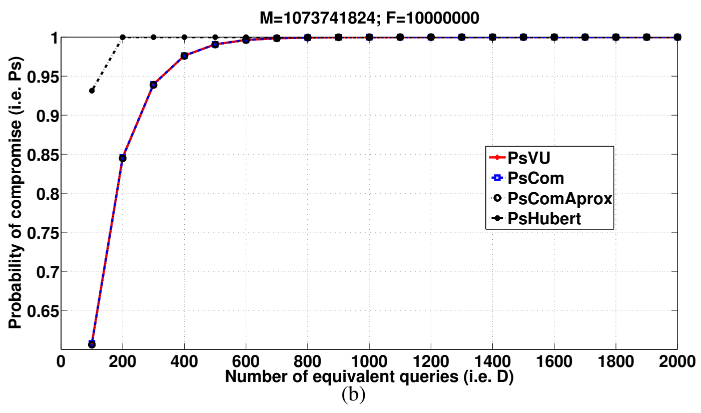


Figure 3. (a) Comparison of the four models with D is fixed at 1,000 outstanding queries (PsVU, PsCom, and PsComAprox are occluded). (b) Comparison of the four models when F is fixed at 10 million packets (PsVU, PsCom, and PsComAprox are occluded).

where $\mathbb{S}(\overline{D}, i)$ is the Stirling number of the second kind [36]. It can be computed according to the following recursive methods:

$$\mathbb{S}(x, y) = \mathbb{S}(x-1, y-1) + y\mathbb{S}(x-1, y), \tag{4}$$

where $\mathbb{S}(0,0) = 1$ and $\mathbb{S}(x,0) = \mathbb{S}(0,y) = 0$ for $x, k \neq 0$.

Since the probability that selecting $\overline{F}$ elements from $(M - i)$ elements is $\frac{\overline{F}!\begin{pmatrix} M-i \\ \overline{F} \end{pmatrix}}{M^{\overline{F}}}$ [1], by combining Eqn.(3), we get:

$$P_f(M, \mathfrak{F}, \mathfrak{D}) =$$

$$\frac{\overline{F}!}{M^{\overline{F}+\overline{D}}} \sum_{i=1}^{\overline{D}} \begin{pmatrix} M-i \\ \overline{F} \end{pmatrix} \begin{pmatrix} M \\ i \end{pmatrix} i!\mathbb{S}(\overline{D}, i) \tag{5}$$

3.2.3 If $D = \overline{D}$ and $F < \overline{F}$

By using similar method as shown in Section (3.2.2), we can easily compute the $P_f(M, \mathfrak{F}, \mathfrak{D})$ in this scenario as follows:

$$P_f(M, \mathfrak{F}, \mathfrak{D}) =$$

$$\frac{\overline{D}!}{M^{\overline{F}+\overline{D}}} \sum_{j=1}^{\overline{F}} \begin{pmatrix} M-j \\ \overline{D} \end{pmatrix} \begin{pmatrix} M \\ j \end{pmatrix} j!\mathbb{S}(\overline{F}, j) \tag{6}$$

3.2.4 If $D < \overline{D}$ and $F < \overline{F}$

By combining the results in Section (3.2.1), (3.2.3), (3.2.2) together, we can obtain the general equation for $P_f(M, \mathfrak{F}, \mathfrak{D})$ as follows:

$$P_f(M, \mathfrak{F}, \mathfrak{D}) =$$

$$\frac{1}{M^{\overline{F}+\overline{D}}} \sum_{i=1}^{\overline{D}} \sum_{j=1}^{\overline{F}} \begin{pmatrix} M-i \\ j \end{pmatrix} \begin{pmatrix} M \\ i \end{pmatrix} i!j!\mathbb{S}(\overline{D}, i)\mathbb{S}(\overline{F}, j) \tag{7}$$

Given the number of spoofed packets sent by an attacker within a window of opportunity, we can compute the probability of compromise (i.e. $P_s(M, \mathfrak{F}, \mathfrak{D}) = 1 - P_f(M, \mathfrak{F}, \mathfrak{D})$), as well as the required data rate (i.e. $R = \frac{FS_{res}}{W}$).

3.3 Comparison to Other Models

There are some existing models such as [2, 37, 40]. The model in [40], (commonly known as the CERT advisory on birthday attacks on DNS servers) could obtain the same result as Eqn.(1). However, the authors in [40] only listed the computation steps instead of giving an exact equation, and did not consider the general case. We rephrase their conceptual model using our symbols as follows:

$$P_f(M, \mathfrak{F}, \mathfrak{D}) = \prod_{k=0}^{\overline{F}-1} (1 - \frac{\overline{D}}{M-k}) \tag{8}$$

To show the improvements offered by our model, we compare the results obtained from the above steps with those derived from our models (i.e. Eqn.(1), Eqn.(2)) and the model suggested in [2], which can be formulated as:

$$P_f(M, \mathfrak{F}, \mathfrak{D}) = 1 - \frac{\overline{DF}}{M} \tag{9}$$

In the following experiments, we set $I = 65,536$, $P = 16,384$ and $N = 1$. We first fix $D = 1,000$ and vary F to

[1]Please note that here we consider selection with replacement. For selection without replacement, the probability is $\frac{\begin{pmatrix} M-i \\ \overline{F} \end{pmatrix}}{\begin{pmatrix} M \\ \overline{F} \end{pmatrix}}$

obtain the results shown in Figure 3(a). Similarly, we then fix $F = 10,000,000$ and vary D resulting in Figure 3(b). The Y-axis in both figures is the probability of compromise and the X-axis indicates the number of spoofed responses (i.e. F) in Figure 3(a), and denotes the number of equivalent queries (i.e. D) in Figure 3(b). In both figures, *PsVU* denotes the result based on the model in [40], *PsCom* denotes the result from Eqn.(1) above, *PsComAprox* denotes the result from Eqn.(2) above, and *PsHubert* illustrates the results obtained from the model in [2]. A ruby implementation of our model is provided in Appendix A for others to experiment with.

Both figures illustrate that the results obtained from the model in [2] deviate from the results of other models. We use the result of the model in [40] as a baseline because it is intuitive. We can easily see that the model in [2] overestimates the results.

PsVU and *PsCom* obtain the same result. The average differences between them observed in the experiments were $1.9414e-014$ and $7.2164e-016$ in Figures 3(a) and 3(b) respectively, due to numerical precision errors. The average difference between *PsComAprox* and *PsCom* were $1.5392e-004$ and $4.6596e-005$ in the two sub-figures respectively. Hence, we regard Eqn.(2)) as a good approximation. We also provide plots of these models in Figure 4 individually.

The author in [37] presented another approximation to Eqn.(1) when $D = F$ as follows:

$$P_f(M, \mathfrak{F}, \mathfrak{D}) \approx (1 - \frac{1}{M})^{\frac{D(D-1)}{2}} \qquad (10)$$

By comparing it with Eqn.(2), it is easy to see that it underestimates the probability of compromise.

4 Analysis

Our model of DNS poisoning is comprehensive and more detailed than previous efforts. To motivate the need for this model, we tracked the rate of DNS poisoning over time, during the critical weeks before and after the August 8, 2008 announcement of a major DNS security flaw. Our study showed numerous instances of poisoning, combined with a large group of resolvers that appear trivially poisonable. This finding motivated the second half of this section, which analyzed interim improvements to DNS resolvers. In the course of our analysis, we identified and patched weaknesses in DNS servers. While falling short of cryptographic improvements to DNS resolution, our approach responds to the clear need for interim solutions and transitional technologies.

4.1 Experiments

We first set out to observe over time the rate at which DNS servers become poisoned. Previous studies [16] have discussed techniques to map large numbers of open recursives and evaluate "incorrect" DNS answers obtained from such hosts (whether done for malicious or commercial gain).

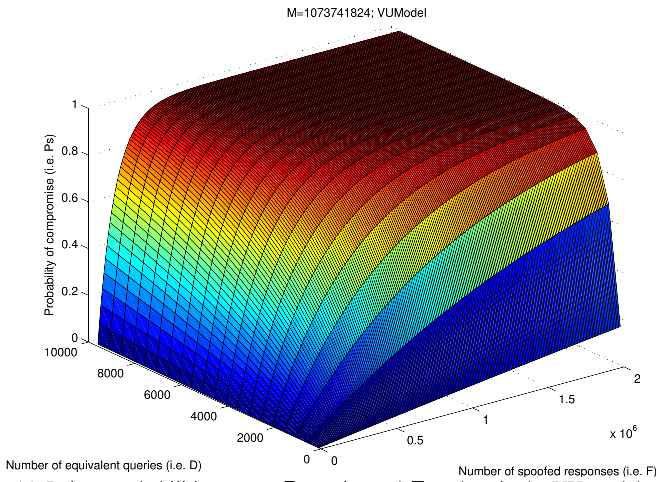
We used this remote monitoring technique as follows: First, we monitored open recursive DNS servers (O-RDNS) for successful poisoning attempts on the 500 most popular domain names [4]. Our monitoring period overlapped the weeks before and after the announcements of the Kaminsky vulnerability [23]. After analyzing the data, we were able to detect successful poisoning attempts and at the same time create a DNS Black List of commonly used malicious IPs used in poisoning. The results showed that attackers were using publicly available DNS poisoning tools without even changing the default IP used by the tool (`1.0.0.0`, `123.123.123.123`, `10.15.8.91`, etc.). Additionally, we observed a significant number of malicious RRsets pointing to networks in China, Japan and in the USA.

The timing of the DNS poisonings with the release of a major DNS vulnerability may of course been coincidental. We therefore estimated the percentage of O-RDNSs servers that applied vendor-recommended patches and implemented SPR.

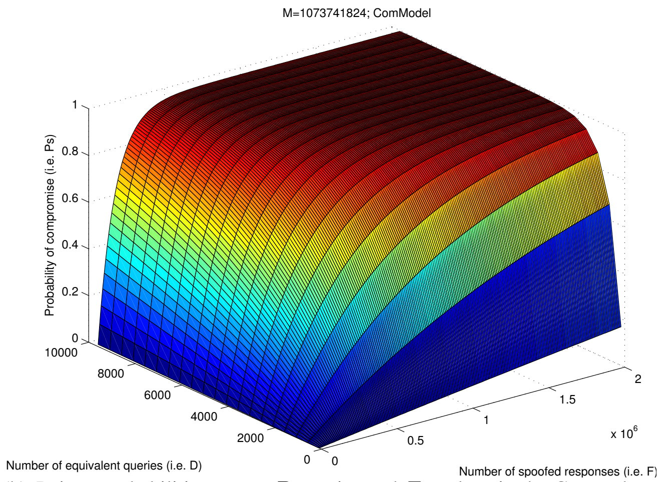
4.2 Successfully Poisoning O-RDNS

We used four scan engines to probe 0.9 million O-RDNS located in 17 different AS in 10 different countries. The scan period was selected to be close to the Black Hat 08 conference. Researcher Dan Kaminsky already had announced that he would make public a very serious DNS vulnerability. Speculations and proof of concept code about the attack were already available in various blogs and underground web sites. These tools were capable and the O-RDNSs were very easy "test" targets for this attack. The probing results from the O-RDNS servers supported our initial worries. In Figure 5, we can clearly observe the increase in the successful poisoning attempts against financial institutions (i.e., `wamu.com`, `bankofamerica.com`) and very popular web sites (i.e., `amazon.com`, `microsoft.com`). Between July 30th and August 18th, we were able to detect a significant number of DNS poisoning attacks against O-RDNS in various ASs. We first selected the 500 most popular domain names [4] and tried to resolve them with the O-RDNS servers identified in previous work [16]. We found more than 200 unique IPs, which were present in reply records across various zones. Several heuristics allowed us to identify poisonous answers in such a large set of data. For example, it is unlikely that two different top-500 websites on the Internet share the same IP address. Yet such a hosting arrangement is commonly used in collocated phishing hosts. Hosts that collided were put onto a list for hand verification. Additionally, we used a voting system to identify unusual RRsets. It is unlikely that an attacker can poison millions of open recursives at the same time. Thus, "unpopular" answers provided by open recursives were similarly flagged for hand verification.

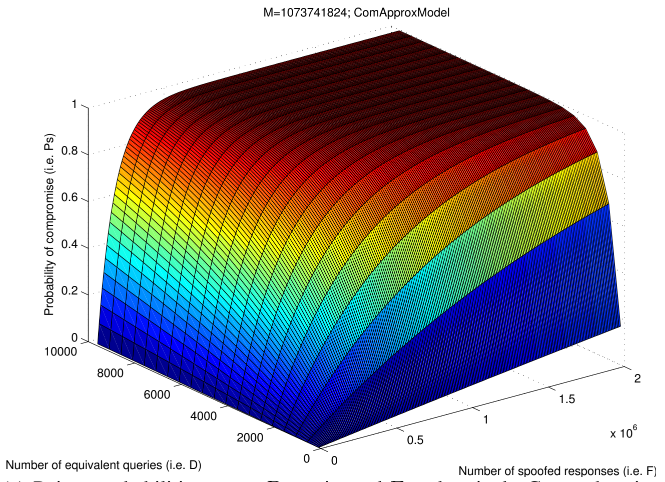
Using an IP to autonomous system number (ASN) map-



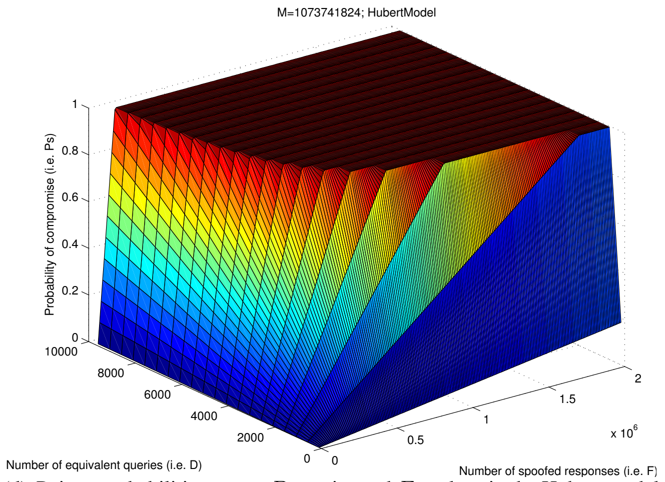
(a) Poison probabilities across D queries and F packets in the VU model.



(b) Poison probabilities across D queries and F packets in the Comprehensive model, Eqn. 1.



(c) Poison probabilities across D queries and F packets in the Comprehensive approximation model, Eqn. 2.



(d) Poison probabilities across D queries and F packets in the Hubert model

Figure 4. Comparison of calculated probabilities between the four models. This figure illustrates the probability of compromise computed through different models. In each subfigure, we calculate the probability under different D (i.e. the number of equivalent queries) and F (i.e. the number of spoofed responses) using one model. (a) shows the results from the VU model; (b) and (c) illustrates the results the results from our model and its approximation model, respectively; (d) presents the results from the Hubert model.

Zone	Successful Attempts	Poisoned Sub-zones	Unique Malicious IPs
amazon.com.	944	4	11
bankofamerica.com.	351	1	25
capitalone.com.	960	3	18
chase.com.	947	2	27
microsoft.com.	827	4	13
icicibank.com.	4416	7	11
wamu.com.	11050	6	24

Table 1. Poisoning evidence in seven popular zones. The first column shows the total number of successful poisonings found in each zone. The second column presents distinct sub-zones that were also poisoned. The last column shows the distinct number of IPs found in poisoned resource records (RRs) in each zone.

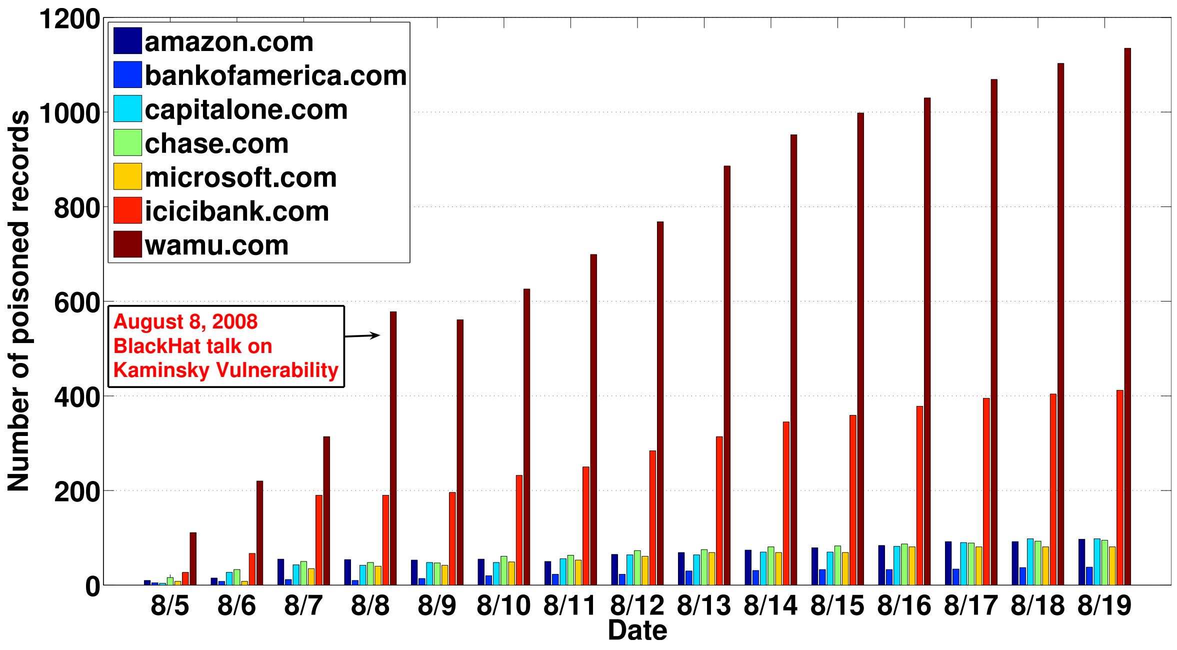


Figure 5. Successful poisoning attempts during the reveal of Kaminsky Class of attack Aug-2008.

ping system [39], we created a white list and black list of IPs. Hosts associated with an organization (e.g., a bank) were found within ARIN allocations given to the organization and announced by the unit's ASN. Our blacklist used the opposite logic: where ASN information could not be associated with the `qname` domain zone in any legitimate way (including hand verification).

In Table 1, we can observe the number of total poisoned records returned from O-RDNS in the first two weeks of August 2008. The majority of the poisoned IPs were located in USA, China and Japan. Figure 5 illustrates the impact of the DNS vulnerability announcement. Prior to the announcement, poisoning against O-RDNS was observed, but steady in rate. In the weeks following the release of automated attack tools, successful attempts against O-RDNS significantly increased.

4.3 Observing the Port Randomization Patch Deployment

Although we could verify each suspected poisoning incident and build a blacklist of malicious IPs, we could not automatically attribute the poisonings to Kaminsky-class attacks. To gain some insights into likely causes of the poisonings, we conducted a scan of O-RDNS for port randomization behavior. Conceptually, one could just requery the hosts repeatedly and measure the standard deviation of port numbers found in answers. We noted the use of a more elegant technique at OARC's self-testing system [45]. A single query is answered with a lengthy `CNAME` (and no offered glue), thereby forcing recursives to requery. Resolution of the `CNAME` requires walking a lengthy delegation chain, each located at a different IP on the multi-homed authority. Thus, using one query, one can induce a tightly packed set follow-up queries from the recursive. This yields dozens of source ports data samples for the cost of merely one outbound open recursive probe.

We implemented a similar technique involving multiple domains under different TLDs. We describe this technique in some detail, since it efficiently generates multiple queries from recursives and consumes a minimum number of authority IPs. Using just three domains, (say, `experiment.biz`, `experiment.org` and `experiment.info`), a series of `CNAME`s forces a query to a different sibling zone. Since we set the TTLs of all records to zero, the recursive is forced to revisit each sibling host. Thus, the initial query for any label under `in.experiment.org` would result in the following `CNAME` storm:

$$\star\texttt{.in.experiment.org} \rightarrow \texttt{a.experiment.biz} \rightarrow \texttt{a.experiment.info} \rightarrow \texttt{a.b.experiment.biz} \rightarrow \texttt{a.b.experiment.biz} \rightarrow \texttt{a.b.experiment.info} \rightarrow \ldots \quad \ldots \rightarrow \texttt{a.b.c.d.e.experiment.biz} \rightarrow \texttt{a.b.c.d.e.experiment.info}: \texttt{IN A 10.0.0.1}$$

This round-robin sibling delegation continues, forcing the recursive to requery at each cut, only to be directed to another sibling. There are limits to how far DNS servers will follow `CNAME` chains, but the zero TTL allows one to repeat this process and generate large numbers of port number samples for a host. A single multi-homed host acts as the authority for all three zones (and all their child zones), so that statistical studies of port randomization are easily obtained from a single capture file. This approach also avoids the need to create customized resolvers to suppress glue (e.g., which could potentially short-circuit recursive delegation walking under the OARC approach.) Thus, we could use robust authority implementations (e.g., a set of optimized BIND installations) to test numerous open recursives simultaneously.

With a high-speed measurement platform in place, we tracked queries from unique O-RDNS by appending a random 9-character label under the top-most `CNAME` (or $\{RND\}$`.in.experiment.org`). A simple wild-card configuration for the first zone ($\star$`.in.experiment.org`) ensures that any query that reaches that zone will be pointed to the next domain name using a CNAME (in this case the: `a.experiment.biz`). This continues until the O-RDNS reaches the `a.b.c.d.e.experiment.info`, where our authoritative name server will respond back to the O-RDNS with a terminating leaf `A` record.

After querying 0.9 million active O-RDNS servers we observed that approximately 57% of the O-RNDs always reply with the same port. A smaller percentage 2% of the O-RDNS used a round robin port switching approach. For the rest of the O-RDNS, 41%, we were not able to find any patterns in their source port selection. A non-zero standard deviation of port numbers suggested the use of random ports within a range (e.g., above port 1024). We therefore assumed such hosts were properly patched for SPR.

Our analysis intentionally stopped short of several other inquiries. For example, we did not attempt to measure the importance of the open recursives, or count the users behind the open recursive. This would have allowed us to categorize the risk posed by each poisoned O-RDNS.

This and related inquires were left for future work or other researchers. Our focus was on demonstrating that (a) many DNS servers are still not patched; and (b) many DNS servers are being poisoned, as shown in previous works [16], using what appear to be new DNS vulnerabilities. These two observations (and the large number of O-RDNS servers at risk) point to the need for interim security improvements. While DNSSEC, DNSCurve, and other proposals may eventually provide cryptographic solutions to poisonings, we documented ample evidence of ongoing attacks to the current DNS infrastructure.

4.4 Survey of DNS Architectures

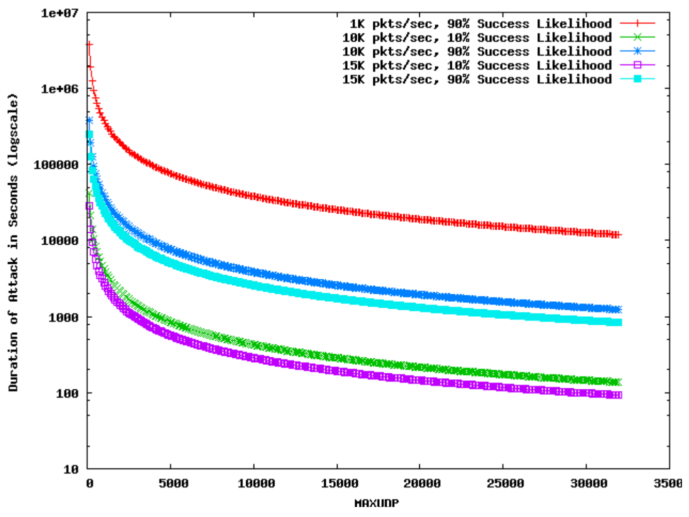
Given the need for interim DNS security improvements, we evaluated how different DNS vendors architected their recursive resolvers and looked for opportunities to measure (using our model) how selective patching can improve security. A comprehensive survey of all DNS tools is difficult, since many are closed-source and even the open source implementations are complex.

In the course of our survey, we also noted that some DNS servers had adopted more aggressive security solutions, but could still benefit from using interim defenses. For example, djbdns's `dnscache` implements efficient source port randomization, but does not implement birthday protection, discussed in [24]. The architect of `dnscache` has observed that, ultimately, DNS resolution based on 16-bit `qid` randomization can be poisoned if attackers send enough spoofed packets [12]. With the recent proposal of DNSCurve, we expect that `dnscache` may soon have an even stronger defense, potentially ending all DNS poisoning risks. We similarly expect DNSSEC, if fully deployed, would also remove forgery from the attacker's arsenal.

We agree with this perspective, but observe the need for short-term protections. We submit that there remains an immediate need to understand interim security improvements. DNS resolvers need birthday protection, DNS-0x20 options, good port randomization, and improved bailiwick handling logic. Our analysis above observed numerous poisoning attacks, and a large population is still at risk for trivial Kaminsky-class attacks. This evidence suggests that the path to cryptographically secure resolution must include these interim defenses.

We therefore worked with the `djbdns` developer community to engineer a patch for `dnscache`[2] to provide birth-

[2]The patch was developed by Jeff King, a PhD student at Georgia Tech and frequent contributor to the djbdns developer community.



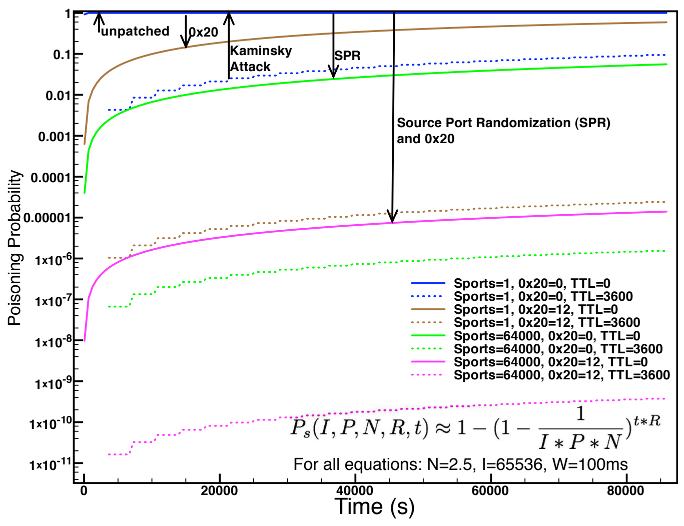


Figure 6. (a) Time-to-poison dnscache without birthday protection. (b) Comparison of various DNS architectural choices, and improvements in security.

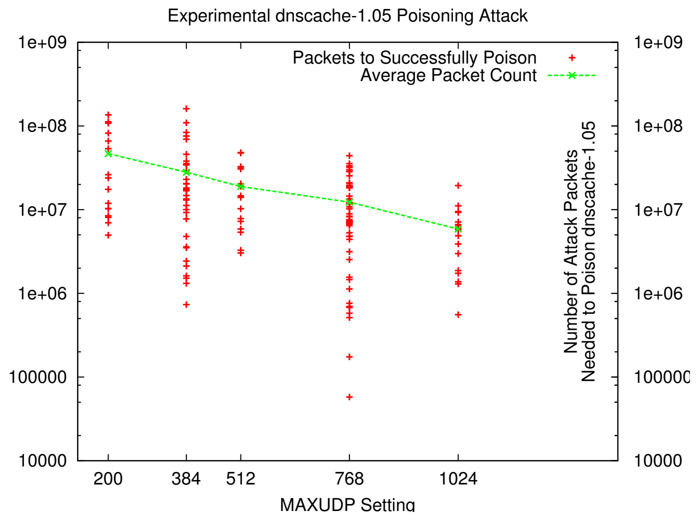


Figure 7. Successful poisoning attacks on `dnscache 1.05`**. Each plot indicates the number of packets required to poison a dnscache instance with a given** `MAXUDP` **setting.**

day protections and made the patch freely available. The patch generally checks the buffer of outstanding queries for any matching instance (based on `qname, type, class, bailiwick`) and "piggybacks" new requests on any old outstanding requests. This removes the birthday window. We have tested the patch in two university networks for several months and found it worked effectively to reduce the risk of attack.

Figure 6(a) shows the improvements realized by applying our patch. Without birthday protection, Figure 6(a) shows it would be possible (under some configurations of dnscache) to poison the resolver in hours or minutes. For example, an attacker sending a large volley of packets (15K q/s) has a 90% chance of success of poisoning a dnscache server configured to allow tens of thousands of simultaneous queries. The attack could completed in minutes. In its default state, `dnscache` ships with a fairly small birthday window—the parameter `MAXUDP` in `dnscache.c` is defined at 200. The dnscache user community, however, sometimes advocates the use of larger values of `MAXUDP` to accommodate more simultaneous queries. Such configurations consequently have a larger birthday window and a significantly higher risk, as illustrated by our model.

To validate the model, we configured a local network running instances of stock `dnscache 1.05`, and altered only the `MAXUDP` variable. We then queried over 5000 times for the authority records for a toy domain, `SOA? abcdefghijkl.mn`. The large volley of initial queries was needed to ensure that no other queries occupied any available slot in the queue. We then spoofed 100 answers, providing `6.6.6.6` as the malicious IPs for NS1 and NS2. This was repeated until success.

We chose to query for authority records because `dnscache` specifically does not cache `SOA` records (as noted in its documentation [11]). This ensured that the test could be restarted quickly and efficiently after a successful poisoning. A parallel monitoring of the dnscache log file would observe acceptance of the malicious answer, and terminate the propagation of additional malicious answers from the attacking host. Because of latency, both in logging and in the poisoning client's kill switch, we may have overcounted the number of packets needed to successfully poison by approximately 1%. (This overcounting provides a more conservative estimate of dnscache's performance.)

We repeated the process to obtain 16 successful poisonings for various `MAXUDP` settings. When `MAXUDP` was set to 512, we were able to poison a stock `dnscache` instance (without a birthday protection patch) in an average of 18,871,764 packets. Note that our attack alternated between spoofing the two authority IP addresses for the toy domain. It's common for authority servers to have two addresses. This doubled the number of attack packets we had to send in our model. (However, for domains served by a single authority IP, the number of malicious packets one must send is half what we report in our experiment.)

When `MAXUDP` was set to 1024 (also a common configuration parameter) we were able to poison dnscache in an average of 5,874,332 packets. Figure 7 shows a plot of the experimental outcomes, and the distribution of packet counts needed to successfully poison dnscache. When `MAXUDP` was set to 2048, we found that dnscache would use significant amounts of CPU time, resulting dropped packets. At approximately 10KPPS, about half the traffic was dropped. This is because dnscache uses list structures to manage open queries, and performs an $O(3N)$ sweep with each response. (By default, the program sets `MAXUDP` to 200, so a list is an appropriate data structure for small query volumes.)

At first, there is nothing surprising about these results. The outcome merely validates what Dan Bernstein has long said about DNS attacks: given enough packets, any DNS server is vulnerable. (Indeed, the difficulty in securing 16-bit resolution systems is what motivated the development of `DNSCurve`). The importance for our work, however, is that these data points match the curve provided by our model.

Our model allows us to evaluate the trade-offs in how one might improve existing DNS architectures, pending a long-term cryptographic solution, as shown in Figure 6(b). With a default `MAXUDP` of 200 (that is, a birthday collision window of 200), `dnscache` has some reduced forgery resistance; however, the use of source port randomization (first innovated in `dnscache`) significantly improves security.

Cryptographic DNS resolution would provide a complete solution. But the history of DNSSEC suggests this process will continue to be slow. In the interim, our model allows one to weigh short-term options for improving security.

5 Related Work

Our work fits into the larger body of literature concerned with DNS poisoning. DNSSEC has been proposed as security extensions of DNS [6–8]. DNSSEC provides origin authentication, data integrity, and validated denial of existence. DNSSEC does not provide confidentiality of data, since the contents of zones can be enumerated. To address this problem (and to provide some efficiencies in delegations), NSEC3 was proposed. [31].

Recently DNSCurve [13] was proposed as an alternative to DNSSEC. Relying on 255-bit elliptic-curve cryptography, DNSCurve provides stronger integrity than DNSSEC's 1024-bit RSA forgery detection. And unlike DNSSEC, DNSCurve does not publish the contents of secured zones, and therefore does not require enumeration of the zone.

Both of these proposals provide cryptographic strength, but only DNSSEC is fully implemented by the world's DNS vendors and is part of the installed based of recursive and authoritative servers. But neither is in wide use. Both will require adoption by the DNS operator community, ISPs, and other networks before becoming widespread.

Numerous short-term solutions have proliferated in this vacuum. DNSSEC Lookaside Validation (DLV) [5] was proposed as a partial implementation of DNSSEC, allowing the designation of single entry points from which the DNS tree can be search for signed delegation.

Other proposals have sought to secure the existing DNS protocol by finding additional sources of entropy. DNS-0x20 [15] uses variations in case formatting of `qnames` to track transactions. Similarly, "Domain Name System (DNS) Cookies" [1] uses additional option (an `OPT RR`) for tracking sessions. These types of proposals are generally distinguished by whether or not they require updates to the recursive and authority server population, or just the recursives.

Randomized UDP source ports are another light weight scheme that only recursives have to implement. First proposed by DJ Bernstein, the practice became wide spread after the announcement of the Kaminsky-class DNS poisoning attack. This approach, while conceptually light weight, can have significant resource costs on busy servers. Servers that implement source port randomization have to potentially perform `select(2)` over large pools of open file descriptors—a resource intensive operation [17]. As a result, source port randomization may not be appropriate for some environments.

The work closest to our current study is A. Hubert's DNS-EXT study of forger resilience [2]. This comprehensive IETF document describes the DNS poisoning problem and provided preliminary mathematical modeling for determining the probability of poisoning. There were several aspects of poisoning considered in our model, which where excluded from the IETF draft. We plan to contribute our model and Appendix A to this effort.

6 Conclusion and Future Work

To aid DNS implementors in judging implementation-security tradeoffs between various interim solutions, we have developed mathematical models to compare interim protection techniques. We provided a complete DNS poisoning model and compared it to previous models. We discussed architectural tradeoffs in managing queries and why certain architectural choices could cause nuanced security concerns. To illustrate these tradeoffs, we described weaknesses in actual implementations and their implications.

Future directions of this work will consider performance, complexity, and verifiability tradeoffs, while adhering to strict security guidelines, in the implementation of DNS servers. We will further consider the development costs associated with various security techniques. Finally, we will further examine how various protection deployments help or hinder the adoption of more complete cryptographic solutions.

Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant No. 0627477 and Grant No. 0831300, and the Department of Homeland Security under Contract No. FA8750-08-2-0141. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation and the Department of Homeland Security.

References

[1] Donald E. Eastlake 3rd. Domain name system (dns) cookies. `http://tools.ietf.org/html/draft-eastlake-dnsext-cookies-03`, 2008.

[2] A. Hubert and R. van Mook. Measures for making DNS more resilient against forged answers. `http://tools.ietf.org/html/draft-ietf-dnsext-forgery-resilience`, 2008.

[3] A. Szmit, M. Tomaszewski, and M. Szmit. Domain name servers's pseudo-random number generators and DNS cache poisoning attack. *Polish Journal of Environmental Studies*, 15(4C), 2006.

[4] Alexa. Alexa the web information company. `http://www.alexa.com/`, 2007.

[5] M. Andrews. The dnssec lookaside validation (dlv) dns resource record, rfc 4431. `http://www.ietf.org/rfc/rfc4431.txt`, 2006.

[6] R. Arends. Protocol modifications for the dns security extensions, rfc 4035. `http://www.ietf.org/rfc/rfc4035.txt`, 2005.

[7] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Dns security introduction and requirements. `http://www.ietf.org/rfc/rfc4033.txt`, March 2005.

[8] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource records for the dns security extensions. `http://www.ietf.org/rfc/rfc4034.txt`, March 2005.

[9] D. Barr. Common dns operational and configuration errors. `http://www.ietf.org/rfc/rfc1912.txt`, 1996.

[10] G. Barwood. Resolver side mitigations draft: dnsext-fr-resolver-mitigations. `http://www.ietf.org`, September 2008.

[11] D. Bernstein. dnscache. http://cr.yp.to/djbdns/dnscache.html, visited in Aug. 2008.

[12] D. J. Bernstein. Dns forgery. `http://cr.yp.to/djbdns/forgery.html`, 2008.

[13] D. J. Bernstein. Introduction to dnscurve. `http://dnscurve.org/`, 2008.

[14] Computer Academic Underground. Cau-ex-2008-0003, bailiwicked_domain.rb. `http://www.caughq.org/exploits/CAU-EX-2008-0003.txt`, 2008.

[15] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased dns forgery resistance through 0x20-bit encoding. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)*, October 2008.

[16] David Dagon, Niels Provos, Christopher P. Lee, and Wenke Lee. Corrupted dns resolution paths: The rise of a malicious resolution authority. In *Proceedings of Network and Distributed Security Symposium (NDSS '08)*, 2008.

[17] David Dagon and Paul Vixie. Setting dns's hair on fire. `http://www.usenix.org/events/sec08/tech/`, 2008.

[18] R. Elz and R. Bush. http://www.faqs.org/rfcs/rfc2181.html, July 1997.

[19] Olafur Gudmundsson and Andrew Sullivan. DNS extensions (dnsext). `http://www.ietf.org/html.charters/dnsext-charter.html`, 2008.

[20] IANA. Port numbers. http://www.iana.org/assignments/port-numbers, August 2008.

[21] infobyte. Isr-evilgrade v1.0.0. http://www.infobyte.com.ar/down/isr-evilgrade-Readme.txt, 2008.

[22] Internet Systems Consortium. Bind 9 administrator reference manual. http://www.isc.org/index.pl?/sw/bind/arm95/, 2008.

[23] Dan Kaminsky. Doxpara research. http://www.doxpara.com, 2008.

[24] Dan Kaminsky. Its the end of the cache as we know it. http://www.doxpara.com/DMK_BO2K8.ppt, 2008.

[25] Amit Klein. BIND 8 DNS cache poisoning. http://www.trusteer.com/docs/bind8dns.html, 2007.

[26] Amit Klein. BIND 9 DNS cache poisoning. http://www.trusteer.com/docs/bind9dns.html, 2007.

[27] Amit Klein. OpenBSD DNS cache poisoning and multiple OS predictable IP ID vulnerability. http://www.trusteer.com/docs/dnsopenbsd.html, 2007.

[28] Amit Klein. Windows DNS cache poisoning. http://www.trusteer.com/docs/microsoftdns.html, 2007.

[29] Amit Klein. PowerDNS recursor DNS cache poisoning. http://www.trusteer.com/docs/powerdnsrecursor.html, 2008.

[30] L. Dorrendorf, Z. Gutterman, B. Pinkas. Cryptanalysis of the windows random number generator. In *Proc. ACM CCS*, 2007.

[31] B. Laurie, G. Sisson, R. Arends, and D. Blacka. Dns security (dnssec) hashed authenticated denial of existence. http://www.ietf.org/rfc/rfc5155.txt, 2008.

[32] P. Mockapetris. Domain names - concepts and facilities. http://www.ietf.org/rfc/rfc1034.txt, 1987.

[33] P. Mockapetris. Domain names - implementation and specification. http://www.ietf.org/rfc/rfc1035.txt, 1987.

[34] T. Nakata. Collision probability for an occupancy problem. *Statistics* & *Probability Letters*, 2008.

[35] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2008.

[36] R. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1997.

[37] J. Stewart. DNS cache poisoning - the next generation. http://www.lurhq.com/dnscache.pdf, 2003.

[38] Joe Stewart. Dns cache poisoning - the next generation. http://www.secureworks.com/research/articles/cachepoisoning, 2002.

[39] Team Cymru. IP to ASN mapping. http://www.team-cymru.org/Services/ip-to-asn.html, 2008.

[40] United States CERT. Various DNS service implementations generate multiple simultaneous queries for the same resource record. VU 457875, November 2002.

[41] US-CERT. Multiple dns implementations vulnerable to cache poisoning. http://www.kb.cert.org/vuls/id/800113, 2008.

[42] US Department of Justice. Eugene e. kashpureff pleaded guilty. http://www.usdoj.gov/criminal/cybercrime/kashpurepr.htm, 1998.

[43] Paul Vixie. DNS complexity, April 2007.

[44] M. Wendl. Collision probability between sets of random variables. *Statistics and Probability Letters*, 64(3), 2003.

[45] Duanne Wessels. Web-based dns randomness test. https://www.dns-oarc.net/oarc/services/dnsentropy, 2008.

A Poison Probability Computation Script – Ruby

The following is an implementation of the comprehensive DNS poisoning model described in Section 3.

```
module CachePoisoning
  class Recursive
    def initialize(ports=4096, recursives=1, ids=2**16, auths=2.5,
      rtt=0.1, pending=1, ox20=false)
      @recursives = recursives # number of recursives
      @ports = ports     # P: number of used ports
      @ids = ids         # I: number of TXIDs
      @auths = auths     # N: number of authoritative name servers
      @rtt = rtt         # W: round trip time between the recursive
                         #    server and the authoritative
      @pending = pending # D: number of simultaneous queries with
                         # the same QNAME birthday protection
                         # effectively sets D=1
      @ox20 = ox20       # O: is 0x20 used (tOgGle CAse LaBEls)
    end
    def probability(attacker, time)
      ex = time / @rtt
      o = (@ox20) ? 2**12 : 1
      i = @ids; p = @ports; n = @auths * @recursives; d = @pending
      m = i*p*n*o
      f = attacker.rate * @rtt
      pf = (1 - (1.0/m))**(d*f)
      pcs = 1 - (( pf )**ex)
    end
  end
  class Attacker
    attr_reader :rate
    def initialize(rate=20E3)
      @rate = rate # rate in packets per second
    end
  end
  class DataGenerator
    def initialize(recursives=[], attackers=[], timemin=0,
      timemax=24*60*60, timedelta=600)
      @recursives = recursives # array of recursive objects
      @attackers = attackers   # array of attacker objects
      @timemin = timemin       # start time
      @timemax = timemax       # end time
      @timedelta = timedelta   # jump
    end
    def generate
      time = @timemin
      data = []
      while time <= @timemax
        timedata = [time]
        @recursives.each do |r|
          @attackers.each do |a|
            timedata << r.probability(a,time)
          end
        end
        data << timedata
        time += @timedelta
      end
      data
    end
    def prettyprint(data)
      data.each do |row|
        puts row.join(" ")
      end
    end
  end
end

if __FILE__ == $0
  bind = CachePoisoning::Recursive.new(16384,1,2**16,1,0.1,1000)
  onegbps = CachePoisoning::Attacker.new(20E3)
  dg = CachePoisoning::DataGenerator.new([bind], [onegbps],
    1, 100, 1)
  data = dg.generate
  dg.prettyprint data
end
```