WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning Attacks

Roberto Perdisci[1,2], Manos Antonakakis[2], Xiapu Luo[2], and Wenke Lee[2]
[1]Damballa, Inc. Atlanta, GA 30308, USA
[2]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA
perdisci@damballa.com, {manos,csxpluo,wenke}@cc.gatech.edu

—

This is an extended version of the DSN-DCCS'09 paper.

Abstract

Recently, a new attack for poisoning the cache of Recursive DNS (RDNS) resolvers was discovered and revealed to the public. In response, major DNS vendors released a patch to their software. However, the released patch does not completely protect DNS servers from cache poisoning attacks in a number of practical scenarios. DNSSEC seems to offer a definitive solution to the vulnerabilities of the DNS protocol, but unfortunately DNSSEC has not yet been widely deployed.

In this paper, we propose Wild-card SECure DNS (WSEC DNS), a novel solution to DNS cache poisoning attacks. WSEC DNS relies on existing properties of the DNS protocol and is based on wild-card domain names. We show that WSEC DNS is able to decrease the probability of success of cache poisoning attacks by several orders of magnitude. That is, with WSEC DNS in place, an attacker has to persistently run a cache poisoning attack for years, before having a non-negligible chance of success. Furthermore, WSEC DNS offers complete backward compatibility to DNS servers that may for any reason decide not to implement it, therefore allowing an incremental large-scale deployment. Contrary to DNSSEC, WSEC DNS is deployable immediately because it does not have the technical and political problems that have so far hampered a large-scale deployment of DNSSEC.

1 Introduction

The normal operation of the Domain Name System (DNS) [21, 22] is vital for a dependable Internet. We *trust* the DNS servers in our every day life to provide us with the correct *domain name* to *IP address* mapping, so that we can browse the Web, send emails, access our bank accounts, etc. Even a partial disruption of DNS may have a catastrophic impact on the Internet.

DNS queries are usually initiated by a *stub-resolver* (e.g., a web browser) on a user's machine, which depends on a *recursive DNS resolver* (RDNS) for obtaining the IP address (or other resources) related to a domain name. The RDNS is responsible for directly contacting the authoritative name servers on behalf or the stub-resolver, cache the response for a given *time to live* (TTL), and forwards it back to the stub-resolver. Since its introduction, DNS has been found to be vulnerable to a number of attacks. In particular, *cache poisoning* attacks have been shown to be quite feasible [25]. Poisoning attacks work by forcing an RDNS to lookup a domain name (e.g., `google.com`), and then sending forged DNS responses back to the RDNS before the "real" valid response from an authoritative name server arrives. Each DNS query contains a 16-bits-long transaction ID (TXID) that allows the RDNS to distinguish valid responses from bogus ones. Therefore, the attacker has to "guess" the correct TXID in order for a forged response to be accepted and stored in the cache. If the attack is successful, the attacker can force the RDNS to resolve the targeted domain name to a malicious IP, and to store the malicious IP in the cache with a long TTL. As a consequence, the next time a stub-resolver queries the RDNS for the same domain name, it will also be redirected to the malicious IP (e.g., users of `google.com` may be redirected to a malicious website that hosts malware or participates in information theft). Recently Kaminsky [15] showed that a successful cache poisoning attack may be accomplished in a matter of seconds, by exploiting a flaw in the DNS protocol.

1.1 Previous Work

The Domain Name System Security Extensions (DNSSEC) have been proposed as a solution to the vulnerabilities of the DNS protocol, and in particular to cache poisoning attacks. DNSSEC adds data origin

authentication and data integrity verification mechanisms to DNS [2, 3, 4, 11]. The implementation and deployment of DNSSEC would therefore provide a robust way of protecting against DNS cache poisoning attacks (as well as other attacks to the DNS) because all the responses are signed and their authenticity can be verified. For example, DNS cache poisoning attacks (as we know them today) would not work because forged responses can be identified and discarded. DNSSEC seems to be the panacea for the vulnerabilities of DNS. Unfortunately, although DNSSEC was proposed back in January 1997 [11], all these years have not been enough for it to be adopted and deployed in a large-scale. The reasons for this are controversial. There seem to be problems related to both technological and "political" issues. From the technological point of view, probably the main obstacle so far has been the use of a NSEC resource record that can allow zone enumeration. This is viewed by many as a security problem for DNSSEC and has only recently been solved with the introduction of NSEC3 [18]. Furthermore, key management for DNSSEC is fairly complex [17]. The main non-technical point of controversy derives from questions such as, "who owns the root name servers?". This is an important question because DNSSEC relies on a chain of trust that depends on signing zones and sub-zones, with signing the root name servers being the principal *anchor of trust*. Of course, different nations around the world have different opinions about who owns the root name servers and is, therefore, entitle to sign their zones. It is not clear when these problems will be solved, and at the moment the deployment of DNSSEC is reduced to a small number of isolated *islands of trust* [24].

Because a large-scale deployment of DNSSEC may not be realized in the near future, a few alternative techniques for protecting against brute-force poisoning attacks have recently been proposed [8, 14, 6, 13, 23]. These techniques are based on the current DNS protocol, and work by increasing the entropy of DNS queries in order to make forging a valid response more difficult. For example, UDP source port randomization [14] was first proposed and implemented by Bernstein in *djbdns* [5], and has been recently implemented by other major RDNS vendors in response to the Kaminsky's attack [10]. In practice, whenever an RDNS issues a DNS query, it selects the source UDP port[1] from which the query is sent at random. This technique significantly increases the hardness of poisoning attacks because now the attacker needs not only to guess the TXID, but also to send the forged responses to the correct UDP port. Unfortunately, UDP source port randomization may not be effective in certain practical scenarios. A significant fraction of RDNS resolvers around the Internet reside behind a load-balancer or firewall devices that implement network address and port translation (NAT/PAT). Many of these devices perform some form of port translation that reduces the randomness of the UDP source ports generated by RDNS resolvers to a guessable (e.g., "round robin-like") use of the ports [10]. Such settings are not uncommon. For example, Leonard et al. showed that 32% of RDNS resolvers in their study use some sort of "hidden" forwarder [19]. A possible solution would be to move the RDNS in front of the NAT/PAT, but this may not be always possible or easy to do, depending on the specific network architecture or configuration (for example, a change in the network architecture may expose the RDNS to other kinds of attacks, beside cache poisoning).

Dagon et al. [8] recently proposed the 0x20-bit encoding, which uses a random combination of lower- and upper-case letters to write domain name queries, and works independently from the presence of NAT/PAT placed in front of RDNS resolvers. Unfortunately, the amount of additional entropy introduced by the 0x20-bit encoding is a function of the length of the queried domain name. For example, for short popular domain names like `hp.com`, `hi5.com`, `cnn.com`, etc., the 0x20-bit encoding only adds 5 or 6 bits of entropy[2]. This makes poisoning attacks a little harder, but surely not infeasible, as we show in Section 3.5. Several other popular domain names from the top 500 global domains according to Alexa (`alexa.com`) contain even less than 6 alphabetic characters that can be used for the 0x20-bit encoding (e.g., `163.com`, `56.com`, `126.com`, etc., for which 0x20-bit encoding offers only 3 additional bits of entropy).

Bernstein recently proposed DNSCurve [6] as an alternative to DNSSEC. DNSCurve uses high-speed elliptic-curve cryptography, and simplifies the key management problem that affects DNSSEC. The main criticism against DNSCurve comes from the fact that no detailed specifications have yet been written (e.g., in the form of Internet drafts or RFCs), and no implementation is currently available to the public. From the limited documentation currently available DNSCurve seems superior to DNSSEC, particularly in terms of key management, however, the latter has been specified in complete detail in several RFCs [2, 3, 4, 11], and has been implemented by most DNS vendors and thoroughly tested. Also, similar to DNSSEC, DNSCurve still requires significant changes at the root and top-level-domain (TLD) name servers and may therefore incur some of the same "political" problems that have so far prevented a large-scale deployment of DNSSEC. Earlier, Bernstein also proposed a technique called *DNS cookies* [7], which makes

[1] Although DNS queries over TCP are possible, the vast majority of queries are transmitted over UDP.

[2] It is worth noting that although these domain names are often queried using the prefix `www.` (e.g., www.msn.com), which would add 3 more bits of entropy, Kaminsky-style attacks [15] can be initiated by queries to non-existent domain names of the type 000.hp.com, 001.hp.com, etc., which do not allow 0x20 to introduce any additional entropy compared to the base domain.

use of wildcards and TXT records to fetch *signed* IP addresses. It is worth noting that although our WSEC DNS solution (discussed below) makes use of wildcard and TXT records, the technique we propose is very different from *DNS cookies*. We use TXT records only to verify if a zone is or is not WSEC-enabled, and use wildcard CNAME records for resolving DNS queries in a *secure* way. We do not propose to sign the response, instead we propose to use a one-time random number (or *nonce*) for each query to distinguish valid answers from bogus ones. Furthermore, to the best of our knowledge, no detailed specifications or publications exist for *DNS cookies*, and no implementation or experimental results are publicly available.

Beside the UDP source port randomization patch [10], 0x20-bit encoding [8], and Berstain's work [6, 7], while working on WSEC DNS we came across the description of an extended query ID (XQID) [13], which shares some similarities with WSEC DNS. It is worth noting, though, that we developed WSEC DNS independently and before the XQID ideas was made public. Also, we would like to emphasize the fact that to the best of our knowledge XQID is only an idea which has not been backed by any scientific study. The impact of XQID on the increase of DNS traffic and latency is not discussed, and no experimental result is provided. Also, contrary to our work, in XQID [13] no algorithm to avoid changes at the root and TLD name servers level is discussed. We discuss XQID and other additional "non-published" related work in Appendix A

1.2 Our contribution: WSEC DNS

In this paper we propose a novel solution to brute-force DNS cache poisoning attacks that is based on increasing the entropy of DNS queries to the point that cache poisoning attacks become practically infeasible. Towards this end, we present Wildcard SECure (WSEC) DNS, a new DNS query process that leverages existing properties of the DNS protocol. Our solution takes advantage of the definition of wildcard domain names given in RFC 1034 [21] and RFC 4592 [20], and of `TXT` resource records [22]. In practice, the basic idea is to wisely use wildcard domain names so that we can prepend a random string to the queried domain names and still obtain a correct answer. These random strings have the effect of significantly increasing the entropy of DNS queries, thus making valid answers difficult to guess. Contrary to 0x20-bit encoding [8], WSEC DNS protects short domain names (e.g., `hp.com`, `hi5.com`, `cnn.com`, etc.) as well as longer ones. Also, WSEC DNS protects against poisoning attacks independently from UDP source port randomization. Therefore, even in those cases when port randomization is made ineffective by the presence of devices such as load balancers or firewalls that perform port translation without preserving randomness [10], WSEC DNS still makes brute-force cache poisoning attacks practically infeasible.

An important advantage of our approach is that DNS operators are not obligated to implement our solution, because WSEC DNS guarantees complete backward compatibility with current name server configurations. However, recursive DNS (RDNS) resolvers that intend to take advantage of the security benefits of WSEC DNS must implement some new functionalities. On the other hand, collaborating DNS operators who want to protect their domain names against possible cache poisoning must be willing to make simple configuration changes to their name servers, by editing their configuration *zone-files* according to the recommendations reported in this paper.

WSEC DNS provides a way to protect RDNS resolvers from brute-force cache poisoning attacks, including Kaminsky's attack [15]. We argue this is very important, considering the serious consequences of successful DNS cache poisoning, and the uncertainty around when DNSSEC will be widely adopted. Unlike DNSSEC, WSEC DNS does not need support from political institutions (which are involved in the process of signing the root and top level domain name servers for DNSSEC). Rather, WSEC DNS only requires collaboration between administrators of RDNS resolvers and authoritative name servers, making incremental deployment possible due to its backward compatibility. We argue that support of incremental deployment is a very important property, because updating all the DNS resolvers and name servers around the Internet at one time is not feasible. We will show that our approach meets all of the following requirements:

- No change to the protocol/format of DNS queries and responses;
- No change at the root and top level domain (TLD) name servers;
- No software change for authoritative name servers;
- Voluntary collaboration of authoritative name servers;
- Complete backward compatibility with current name servers' configurations;
- Support of incremental deployment;
- Independence from the architecture/configuration of the network where RDNS reside;
- Brute-force cache poisoning attacks are practically infeasible;
- Transparent to users.

2 Background and Threat Model

In this section we provide a brief description of DNS cache poisoning attacks. Due to space constraints we are not able to provide a complete introduction to the DNS protocol here. We therefore assume the reader is already familiar with DNS concepts and terminologies. If this is not the case, we recommend the reader to refer to [21, 22] for

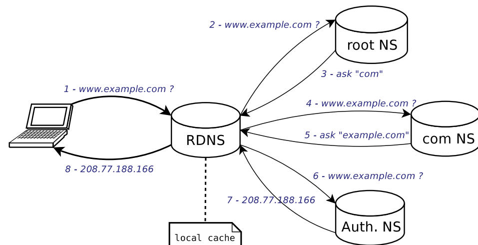

Figure 1: DNS Query Resolution.

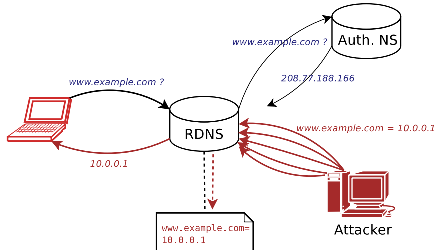

Figure 2: Example of DNS Cache Poisoning Attack Scenario. Interactions with root and TLD nameservers are omitted for simplicity.

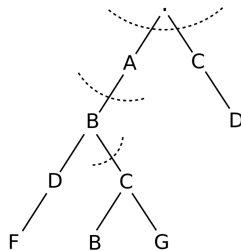

Figure 3: Example of domain name space and zone cuts (identified by dashed curves).

further details.

2.1 DNS Concepts and Terminology

The domain name space is structured like a tree. Each node and leaf on the tree corresponds to a resource set [21]. A domain name identifies a node in the tree. For example, the domain name `F.D.B.A.` identifies the path from the root "." to a node `F` in the tree (see Figure 3). It is worth noting that two *brother* nodes cannot share the same name, but a *child* node can have the same name as its *parent*. The set of resource information associated with a particular name is composed of resource records (RRs). For example, `F.D.B.A.` may be associated with one or more RRs of type `A` containing an address in IPv4 format, one RRs of type `TXT` containing a text description of the domain name, etc. The domain name related to a given RR is called the *owner* of that RR [21]. The depth of a node in the tree is sometimes referred to as *domain level*. For example, `A.` is a top-level domain (TLD), `B.A.` is a second-level domain (2LD), `D.B.A.` is a third-level domain (3LD), and so on.

Zone Authority. The information related to the domain name space is stored in a distributed *domain name database*. The domain name database is partitioned by "cuts" made in the name space between adjacent nodes. After all cuts are made, each group of connected nodes represent a separate *zone* [21]. Each zone has at least one node, and hence a domain name, for which it is authoritative. For each zone, a node which is closer to the root than any other node in the zone can be identified. The name of this node is often used to identify the zone. For example, assume there is a zone cut in the path betwen nodes `B` and `A` in the path `F.D.B.A.` (see Figure 3). In this case, the zone of `F.D.B.A.` is usually identified with the domain name `B.A.` (the last ".", which represets the root, is often omitted). The RRs of the nodes of a given zone are stored in one or more name servers. A given name server will typically support one or more zones. A name server that has complete knowledge about a zone (i.e., stores the RRs for all the nodes in the zone) is said to have *authority* on that zone [21]. A name server can delegate the authority over part of a (sub-)zone to another nameserver.

Wildcard Domain Names. A wildcard domain name is a domain name having its initial (i.e., leftmost or least significant) label be the "*" character [20]. For example `*.www.example.com` is a wildcard domain, where "*" is interpreted as "any valid combination of characters". On the other hand, `www.*.example.com` is not a wildcard domain (the "*" between `www` and `example` is not interpreted as "any valid combination of characters", but represents the single "*" character itself). Assume the name server that has authority on the `example.com` zone is configured to accept wildcard domains of the kind `*.www.example.com` and return the IP address (if a RR of type `A` is queried) `208.77.188.166`. In this case, if we query for the IP address `<random_string>.www.example.com`, this domain will match the wildcard domain `*.www.example.com`, and the ANS will respond with `208.77.188.166`.

2.2 DNS Query Resolution

DNS queries are usually initiated by a *stub-resolver* (e.g., a web browser) on a user's machine, which relies on a *recursive DNS resolver* (RDNS) for obtaining a set of *resource records* (RRs) owned by a given domain name. The RDNS is responsible for directly contacting the authoritative name servers on behalf or the stub-resolver, obtain the requested information, and froward it back to the stub-resolver. The RDNS is also responsible for caching the obtained information for a certain period of time, called *Time To Live* (TTL), so that the if the same or another stub-resolver queries again for the same information within the

TTL time window, the RDNS will not need to contact the authoritative name servers (thus obviously improving efficiency).

Consider the scenario in Figure 1. The stub-resolver requests the IP address of `www.example.com` to the RDNS. First, the RDNS checks its local cache to see if it already knows the answer, and if the cached information is not expired. If the requested information is not in the cache, the RDNS will try to retrieve the IP address of the *authoritative name servers* (ANS) for `www.example.com`. If no ANS is found in the cache, the RDNS will look for the IP of the ANS for `example.com`, then the ASN for the *top level domain* (TLD) `com`, and finally the root name servers. Assume only the IP of the root name servers is in the cache. At this point the RDNS will ask the root name servers "what is the IP of `www.example.com`?". The root name servers will respond with "I don't know, but you can ask the name server which have authority on the `com` zone. Here are their IP addresses". The RDNS will then contact the `com` name servers and ask "what is the IP of `www.example.com`?". Again, the `com` name servers will not know the answer and will respond with "ask the name server which have authority on the `example.com` zone. Here are their IP addresses". Finally, the RDNS will contact the ANS of the zone `example.com` and obtain the requested IP address.

During the process of discovering the IP address of `www.example.com`, the RDNS will store all the information it receives into its local cache. This includes the IP addresses of the ANSs for the `com` zone, the IP addresses of the ANSs for the `example.com` zone, and the IP address of `www.example.com`. Each cache entry will have an associated Time To Live (TTL), after which the cache entry expires and the RDNS needs to query for that information again. Now, assume after a while another stub-resolver queries for the IP address of `www.example.com`. If the cache entry that contains the IP address of `www.example.com` has not expired, the RDNS will immediately respond without the need to contact any external ANS. On the other hand, if the cache entry that stored the IP address of `www.example.com` has expired, the RDNS needs to reinitiate an interactive query process. At this point, the RDNS knows that it can directly ask that question to the ANS for the `example.com`, because the RDNS has the IP address of that ANS in the local cache (assuming the related cache entry has not expired, yet). Therefore, the RDNS does not need to contact the root and `com` name servers.

2.3 DNS Cache Poisoning

DNS queries are transmitted over UDP[1]. Each DNS query contains the following information: 1) a transaction ID (TXID); 2) the queried domain name; 3) and the requested resource record (RR) class and type. In order to identify a valid answer to a DNS query, the following information should be verified: a) the response packet needs to come from the same IP the query was sent to; b) the source UDP port used to send the query must match the port on which the response is received; c) the TXID in the answer must match the TXID in the response; d) the domain name reported in the question section of the response must match the queried domain name. We assume the transaction ID (TXID) and UDP source port used by the RDNS when issuing DNS queries are chosen at random.

However, it is worth noting that until recently the UDP source port used by popular RDNS software like BIND 9 and Windows DNS server was easy to predict [16], and even the TXTID was not very well randomized [16, 25]. DNS cache poisoning attacks exploit the low entropy of DNS queries (due mainly to a poor randomization of TXID and UDP source port), which makes valid answers predictable [16, 25].

A traditional brute-force DNS cache poisoning attack scenario is as follows (see Figure 2). Assume an attacker tries to poison the IP address of `www.example.com`. The attacker first sends a query for `www.example.com` to the RDNS, thus forcing it to initiate the recursive query process and interact with the authoritative name servers. If the attacker is able to guess the TXID and source UDP port, and send well-crafted (spoofed) response packets to the RDNS before the legitimate answer from the real authoritative name server is received, the DNS poisoning attack will be successful. This attack works because the RDNS will accept the first valid answer it receives. As a result, it will store the IP address (or other RR information) that the attacker sent in the positive cache for the entire *time to live* (TTL) chosen by the attacker. The consequence is that all clients using that particular RDNS server will subsequently receive the attacker's IP address every time they query for `www.example.com`. For example, every time the users want to visit a web page on that domain, they may be redirected to the attacker's malicious website. This may expose the users to a variety of attacks such as information theft or malware infection. If the attacker is not able to forge a valid answer before the real valid answer arrives at the RDNS, the attacker will need to wait until the related genuine cache entry for the IP address of `www.example.com` expires before retrying.

2.4 Kaminsky's Attack

Recently, Dan Kaminsky discovered a flaw in the DNS protocol [15] that allows an attacker to perform successful cache poisoning attacks with little effort, compared to the traditional poisoning attack.

Assume an attacker wants to poison the cache of the RDNS for `www.example.com` zone. Kaminsky's attack works as follows:

1. We assume the attacker has control of a stub-resolver behind the RDNS (this is not a strong assumption if we consider that simply soliciting a legitimate user to visit a malicious web-page may turn the browser into the perfect stub-resolver under control of the attacker). The attacker's stub-resolver initiates a query for the IP address of a non-existent domain, say `001.example.com`.
2. The RDNS will therefore contact one of the authoritative name servers for the zone `example.com`. Without loss of generality, assume there is only one name server that has authority on the `example.com` zone, say `a.iana-servers.net.`, and that the IP address for such name server is `192.0.34.43`.
3. At this point the attacker will send lots of spoofed responses to the RDNS, which will pretend to come from `192.0.34.43` and try to guess the correct transaction ID and UDP port. The attacker's spoofed responses all say "I do not know the IP of `001.example.com`, but I can tell you who can give it to you". The "who can give it to you" translates into an authority section of the response containing `www.example.com`, and the additional section containing `6.6.6.6` as a "glue" record [22], where `6.6.6.6` is an IP address under control of the attacker.
4. The RDNS will update its cache with such information, namely the fact that `www.example.com` has IP address `6.6.6.6`, and therefore the attack succeeds.

[1] describes a variant of the Kaminsky's attack that makes use of `CNAME` RRs. The greatest advantage of Kaminsky-style attacks is that if the poisoning attack does not succeed at the first trial, the attacker can immediately retry querying for another non-existent domain in the same zone, say `002.example.com`, `003.example.com`, etc., without need to wait for the TTL of the domain the attacker wants to poisoning to expire. This greatly improves the probability of success for the attacker, as we show in the analysis presented in Section 3.5.

It is important to note that Kaminsky's attack is another way of performing brute-force cache poisoning against the RDNS. Unlike "traditional" poisoning attacks, the greatest advantage of Kaminsky's attack is that if the poisoning attack does not succeed at the first trial, the attacker can immediately retry, without the need to wait for the TTL of the genuine cache entry to expire. This greatly improves the probability of success of the attack in a given time window, as we show in Section 3.5.

2.5 Threat Model

In this work we address brute-force DNS cache poisoning attacks, both traditional and Kaminsky's style. Similarly to previous work [14, 8], we assume the attacker does not have visibility of the DNS traffic flowing from the RDNS to the legitimate authoritative name server (if the attacker had such visibility there would be no need to launch traditional or Kaminsky's style poisoning attacks, because there would be nothing to guess and only one forged packet would be sufficient to poison the cache). At the same time, we assume the attacker is able to forge DNS responses and send them to the RDNS using a very large bandwidth

3 WSEC DNS

RFC 1034 requires that valid responses to DNS queries must report a copy of the queried domain name in the question section. We propose to take advantage of this simple property of the DNS protocol, and to leverage the use of wildcard domain names as defined in RFC 1034 and RFC 4592 to increase the entropy of DNS queries in order to make brute-force cache poisoning attacks practically infeasible. A wildcard domain name is a domain name having its initial (i.e., leftmost or least significant) label be the "*" character [20]. For example `*.www.example.com` is a wildcard domain, where "*" is interpreted as "any valid combination of characters".

3.1 Configuration Requirements

WSEC DNS requires no software change for name servers. However, DNS operators who want to benefit from the security properties of the WSEC DNS query process must apply simple configuration changes to their *zone-files*. A zone-file is a configuration file used by a name server to load information about a *zone*[3] (e.g., `example.com`), and its subdomains (e.g., `www.example.com`, `mail.example.com`, etc.). The information carried by zone-files is written in terms of resource records (RR). For example, RRs of type `A` are used to report IP addresses in IPv4 format, `CNAME` RRs are used for name aliasing, `TXT` records are used to carry descriptive information about a domain name.

Figure 4 shows an example *zone-file* that describes the content (i.e., sub-domains and resource records) of the zone `example.com`[4]. Consider the configuration line:

```
www  IN  A  10.0.1.6
```

This tells the name server that the domain name `www.example.com` *owns* the IP address `10.0.0.6`.

[3] a zone can be defined as a portion, or sub-tree, of the tree-structured domain name database [21].

[4] Notice that the configuration lines marked with the comment "`; WSEC`" in Figure 4 are the ones that have been added to the zone-file in order to enable WSEC DNS queries. It is also worth notice that the additional configuration lines needed in order to enable WSEC DNS queries can be generated automatically by writing a simple script which takes the original zone file in input and follows the steps described above to produce the new WSEC-compliant zone file.

```
$ORIGIN example.com.
@       IN      SOA     ns1.example.com    hm.example.com. (
                        2001062502 ; serial
                        21600      ; refresh after 6 hours
                        3600       ; retry after 1 hour
                        604800     ; expire after 1 week
                        600 )      ; minimum TTL 10 minutes
        IN      NS      ns1.example.com.
        IN      NS      ns2.example.com.
        IN      TXT     "v=spf1 a mx -all"
        IN      MX      10      mail.example.com.
        IN      A       10.0.1.1
*       IN      A       10.0.1.100
ns1     IN      A       10.0.1.2
ns2     IN      A       10.0.1.3
mail    IN      A       10.0.1.4
www     IN      A       10.0.1.6
www     IN      AAAA    2001:db8::3
www     IN      TXT     "This is our website"
*.web   IN      A       10.0.1.7
ftp     IN      CNAME   www
;
;
;; RRs added for enabling WSEC DNS are reported below
;
*       86400   IN    TXT   "|wsecdns=enabled|" ; WSEC
*.web   86400   IN    TXT   "|wsecdns=enabled|" ; WSEC
_test_._wsecdns_  86400  IN    TXT   "|wsecdns=enabled|" ; WSEC
;
*._wsecdns_       IN   CNAME   example.com. ; WSEC
*._wsecdns_.ns1   IN   CNAME   ns1  ; WSEC
*._wsecdns_.ns2   IN   CNAME   ns2  ; WSEC
*._wsecdns_.mail  IN   CNAME   mail ; WSEC
*._wsecdns_.www   IN   CNAME   www  ; WSEC
*._wsecdns_.ftp   IN   CNAME   ftp  ; WSEC
;
*._test_._wsecdns_      IN   CNAME   _test_._wsecdns_ ; WSEC
*._test_._wsecdns_.ns1  IN   CNAME   _test_._wsecdns_ ; WSEC
*._test_._wsecdns_.ns2  IN   CNAME   _test_._wsecdns_ ; WSEC
*._test_._wsecdns_.ftp  IN   CNAME   _test_._wsecdns_ ; WSEC
*._test_._wsecdns_.www  IN   CNAME   _test_._wsecdns_ ; WSEC
```

Figure 4: Example of WSEC-compliant zone file written using the syntax for BIND 9.5 (www.isc.org) name servers.

In order to enable WSEC DNS queries for a given zone (e.g., the zone `example.com` and all its sub-domains), the following modifications need to be applied to its related zone-file:

a) Add two `TXT` resource records (RRs) that contain information about whether *WSEC queries* are supported or not for the domains in that zone. For example, in Figure 4 the lines:

```
*                  86400 IN TXT |wsecdns=enabled|
_test_._wsecdns_ 86400 IN TXT |wsecdns=enabled|
```

have been added for this purpose. The string `|wsecdns=enabled|` is used to communicate the fact that the zone `example.com` supports WSEC DNS queries (the "_" characters in `_test_._wsecdns_` are used to avoid collisions with other pre-existent names in the zone).

b) Two `CNAME` RRs are introduced for each non-wildcard domain name in the zone,. For example (see Figure 4), if the original zone-file contains an RR of any type for `www.example.com`, the WSEC-compliant version of the zone-file must contain two additional lines of the type:

```
*._wsecdns_.www IN CNAME www
*._test_.wsecdns_.www IN CNAME _test_.wsecdns_
```

which say that `*._wsecdns_.www.example.com` is an alias for `www.example.com`, and

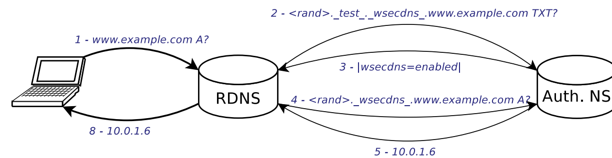

Figure 5: Simplified view of WSEC DNS query process.

`*._test_._wsecdns_.www.example.com` is an alias for `_test_._wsecdns_.example.com`. We informally refer to these configuration lines as *wildcard* `CNAME` *records*.

c) A `TXT` RR is introduced, for each wildcard domain, excluding wildcard `CNAME` RRs. For example, in Figure 4 we introduced the following line `*.web IN TXT "|wsecdns=enabled|"` because a wildcard `A` RR for `*.web.example.com` was present in the original zone-file. We informally refer to these configuration lines as *wildcard* `TXT` *records*.

3.2 The WSEC DNS Query Process

A simplified view of the WSEC DNS query process is represented in Figure 5. Assume a host queries for the RR of type `A` (i.e., the IP address) owned by `www.example.com`. At this point the RDNS generates a random alphanumeric string `<rand>` of length N, and queries for the `TXT` RR of `<rand>._test_._wsecdns_.www.example.com`. This query is used to perform a sort of "handshake" with the authoritative name server (ANS) for the domains in the `example.com` zone. If the ANS for the zone `example.com` has enabled WSEC queries (i.e., the zone-file for the zone `example.com` on the ANS respects the configuration requirements discussed in Section 3.1), it will report the fact that `<rand>._test_._wsecdns_.www.example.com` is an alias for `_test_._wsecdns_.example.com` and that a `TXT` RR exists for this domain, which is equal to the string `"|wsecdns=enabled|"`. At this point, WSEC DNS queries are considered enabled, and instead of querying for `www.example.com A?`[5], the RDNS will query for `<rand>._wsecdns_.www.example.com A?`, i.e., the original query from the host to the RDNS plus the string `<rand>._wsecdns_.` prepended.

Since the ANS is WSEC-enabled, this query will match the wildcard "`*._wsecdns_.www IN CNAME www`" (see zone-file in Figure 4), which says that `*._wsecdns_.www.example.com` is an alias for `www.example.com`. Because the ANS is authoritative for

[5]For the sake of simplicity, in the following we will use the notation `<domain name> <Resource Record Type>?` to mean "query for the resource records of type `<Resource Record Type>` owned by `<domain name>`". Resource Records of type `A` are used to translate a domain name into a set of IPv4 addresses [22].

www.example.com, according to RFC 1034 [21] the response from the ANS (in this example) will include: 1) the queried domain name <rand>._wsecdns_.www.example.com in the question section; 2) an CNAME RR that says that <rand>._wsecdns_.www.example.com is an alias for www.example.com; and 3) an A RR that reports the IP address of www.example.com, which is the information originally requested by the stub-resolver. The RDNS will first make sure that the domain name in the question section of the response actually matches the queried domain name (including the random prefix <rand>). If a mismatch occurs, the response will be discarded. Finally, the RDNS will *normalize* the answer received from the ANS by cutting the CNAME entry that starts with <rand>._wsecdns_. from the response, caching the normalized response, and forwarding it to the host (a more detailed description of the WSEC response normalization process is reported in the Appendix).

It is worth noting that corporate networks often use chains of DNS resolvers and forwarders. In order to avoid multiple WSEC prefixes to be added to a DNS query, the simplest way to deploy WSEC DNS is to only enable WSEC queries at the last RDNS resolver (i.e., the closest one to the edge of the network). In some corporate networks, a DNS forwarder with caching abilities may be present in front of one or more RDNS servers. Therefore, when adding wildcard CNAME RRs to a zone in order to enable WSEC queries, ANS administrators should set the TTL of those records to zero. This will prevent the forwarders from caching wildcard CNAME RRs used in WSEC queries. Because of the uniqueness of the WSEC random prefix, caching such RRs is useless, and may cause an unnecessary growth of the forwarder's cache.

It is easy to see that the basic version of the WSEC DNS query process described above has the side effect of doubling the DNS traffic from the RDNS to the name servers. We will show in Section 3.3 that we can make use of the same cache system used by "standard" RDNS implementations in order to greatly reduce the amount of additional traffic resulting from the WSEC DNS query process.

Security Benefits WSEC DNS protects the RDNS's cache from poisoning attacks against the domain names (e.g., www.example.com) in WSEC-enabled zones (e.g., example.com), including Kaminsky's attack. This is because the attacker is now required to make a much bigger effort to try to guess the random string <rand> used as prefix in WSEC queries, in addition to the random transaction ID (and possibly source UDP port). If the attacker is not able to forge packets with the correct <rand> combination and send them to the RDNS before the genuine authoritative response arrives from the ANS, the attack will fail. This holds true for all the steps of the WSEC DNS query process, regardless of the requested type of RR in the original queries from the hosts, because a new random prefix <rand> is issued with every query sent by a WSEC-compliant RDNS to the name servers, including during the interaction with the root and TLD name servers. We will show in Section 3.5 that using uniformly random strings of only five alphanumeric characters, WSEC DNS makes Kaminsky's attack practically infeasible even for very motivated and powerful attackers.

Transparency Property It is easy to see that the WSEC DNS query process is completely transparent to the host, thanks to the WSEC response normalization algorithm described above. The user will always see the correct response to the original query as if the RDNS was using the "standard" DNS query process, instead of the WSEC DNS process. It is worth noting that queries for resource records other than A, (e.g., AAAA, CNAME, MX, TXT, NS, etc.) can be handled by WSEC DNS. The WSEC DNS query process works independently from the RR type requested in the original query from the host, thanks to the use of CNAME wildcards added in order to make a zone WSEC-enabled as described in Section 3.1.

Backward Compatibility In the cases when a zone is not WSEC-enabled, the ANS will answer to the WSEC "handshake" (i.e., the first query <rand>._test_._wsecdns_.www.example.com TXT?) with NXDOMAIN (non-existent), or possibly with a string that does not contain "|wsecdns=enabled|". In this case the RDNS will simply issue the second query using the original query www.example.com A? from the host (without the WSEC random prefix), and forward the response to the host. A *negative WSEC cache* is used to avoid to retry over and over a WSEC "handshake" with zones that are not WSEC-enabled, as explained in Section 3.3. Only after the negative WSEC cache entry expires (e.g., after one day), the RDNS will retry the WSEC handshake. The backward compatibility allows WSEC DNS to be deployed incrementally, thus making our solution practical.

WSEC TXT TXT resource records were originally meant for storing descriptive text about domain names [22], but are now widely used to carry information related to the Sender Policy Framework (SPF) [26] to mitigate the spam emails phenomenon. We would like to emphasize the fact that wildcard TXT records introduced for enabling WSEC DNS queries do not interfere with either SPF or simple descriptive text. Because of limited space, we discuss the details of how WSEC DNS and SPF can easily coexist in Appendix B. Furthermore, in Appendix B we discuss how the WSEC DNS query process can handle those corner cases in which introducing wildcard TXT RR for enabling WSEC

DNS queries may create conflicts with preexisting wildcard CNAME RR in a zone-file.

3.3 Positive and Negative WSEC Caching

Implementing the WSEC DNS query process without a *WSEC caching system* would have the side effect of doubling the volume of DNS traffic on the Internet and the average latency between users' DNS queries and the related answer, due to the fact that the "handshake" between the RDNS and ANS would have to be repeated for each query. In order to solve these problems, we can take advantage of the concepts of positive and negative cache, as explained in the following. The RDNS resolvers that intend to use WSEC DNS must implement a positive and negative cache that stores information about the zones that are or are not WSEC-enabled (i.e., whose zone-file does or does not follow the configuration requirements described in Section 3.1), respectively. We will describe the WSEC positive cache first, and afterwards we will describe the WSEC negative cache. For the sake of simplicity, we assume here that if a certain zone is configured according to WSEC specifications, all its sub-zones will also be configured to support WSEC DNS. This restriction can be easily relaxed, as explained in Appendix C.

An entry of the WSEC positive cache should contain the following information: 1) a zone name; and 2) a *time to live* (TTL), after which a TXT query for the WSEC handshake needs to be reissued. This information is conveyed by a TXT resource record such as:
test._wsecdns_ 86400 IN TXT "|wsecdns=enabled|"
for the zone we are considering, as we explained in Section 3.1. For example, the zone example.com in Figure 4 has a TXT record of this kind, with a TTL in seconds equal to 86400 seconds (i.e. one day). Therefore, the WSEC positive cache entry would be "example.com wsecdns=enabled 86400". Once a zone has been stored in the positive WSEC cache, the RDNS will not perform the WSEC handshake for domains in that zone, until the TTL of the positive WSEC cache expires. Therefore, assuming the zone example.com is in the positive WSEC cache, the next time a host queries for a domain in that zone, say mail.example.com A?, because example.com is in the positive WSEC cache the only query issued by the RDNS will look like <rand>._wsecdns_.mail.example.com A?. Therefore, the positive cache limits the increase in DNS traffic due to the WSEC handshake (see step 2 in Figure 5), because the WSEC handshake has to be attempted only once a day for a given zone.

Because WSEC DNS does not require every authoritative name server (ANS) on the Internet and its zones to be WSEC-enabled (some ANSs may decide not to increase protection of their domains against poisoning attacks at the cost of editing their zone-files), some zones may never be present in the positive WSEC cache. Therefore, this would cause the RDNS to initiate the WSEC hadshake using TXT queries, thus doubling the traffic towards non-WSEC-enabled ANS. To avoid this problem, RDNS resolvers that implement WSEC DNS are required to also implement a negative WSEC cache. Each entry in the WSEC negative cache should contain two pieces of information: 1) the name of the non-WSEC-enabled zone; and 2) the negative entry TTL. For example, assume the zone vulnerabledns.com decides not to follow the WSEC DNS configuration recommendations described in Section 3.1. The first time a user queries for a domain in that zone, say www.vulnerabledns.com, the RDNS will issue a <rand>._test_._wsecdns_.www.vulnerabledns.com TXT? query, in order to perform the WSEC handshake for the zone. This query will not match any entry in the zone-file for vulnerabledns.com, and the string |wsecdns=enabled| will not be received by the RDNS. Therefore, the RDNS will store the zone vulnerabledns.com in the negative WSEC cache (only if the RDNS received an authoritative response that did not come directly from a root or TLD name server, otherwise it means that the domain is not *registered* and the RDNS will simply returned NXDOMAIN to the stub-resolver), and will simply issue the original www.vulnerabledns.com A? query received from the host. At this point, the RDNS will not issue a WSEC handshake (a TXT query) towards any domain in the vulnerabledns.com zone until the TTL of the negative cache entry expires. The TTL for the entries in the negative WSEC cache is a configuration parameter for the RDNS, and it is recommended to be at least several hours.

3.4 Protecting TLD NS Entries

When querying for a domain name in a WSEC-enabled zone, besides protecting the RDNS's cache from poisoning attacks against the queried domain, e.g., www.example.com, WSEC DNS naturally protects the RDNS from poisoning of the top level domain (TLD) name servers. The reason is that the random WSEC prefix (see Section 3.2) will always be present during the query resolution process for www.example.com, including during the interaction between the RDNS and the root and TLD name servers for discovering the authoritative name server (ANS) for the queried domain. This protection mechanism does not require the root and TLD name servers to explicitly enable WSEC DNS queries themselves.

An attacker may also try to poison the IP address of TLD name servers in the RDNS's cache directly. As an example, assume the attacker intends to poison the cache of an RDNS by injecting a malicious address for the name

server A.GTLD-SERVERS.NET, which is a delegation-only name server for the .com TLD. We would like to emphasize the fact that if the zone GTLD-SERVERS.NET was WSEC-enabled, the attack would not succeed, because the queries initiated by the attacker would be "protected" by the WSEC prefix <rand>._wsecdns_.. However, we understand that requiring changes in the configuration of TLD name servers may encounter the resistance of the Internet community, given the special role of such servers (requiring changes at the root and TLD name servers is one of the main reasons that have kept DNSSEC from being deployed and adopted on a large-scale). However, it is possible to protect TLD name servers from poisoning with neither configuration nor software changes at the root and TLD name servers levels. The solution we propose involves the application of a *secure* TLD cache update policy for RDNS resolvers in combination with the use of WSEC random prefixes. For example, even if the zone GTLD-SERVERS.NET is not WSEC-enabled, we can defend A.GTLD-SERVERS.NET by noticing that the RDNS's "standard" cache will contain two pieces of information about the domains in that zone: 1) A.GTLD-SERVERS.NET is a name servers for the com TLD; and 2) A.GTLD-SERVERS.NET's IP address.

As *legitimate* changes in the IP address of TLD name servers are very rare, if the IP address of a TLD name server in the RDNS's cache is about to be overwritten with a new address (e.g., because of a poisoning attack), the RDNS can simply verify the received information by performing an additional query to one of the root name servers[6]. For example, before overwriting the IP address of A.GTLD-SERVERS.NET in the cache, the RDNS will query one of the root name servers for <rand>._wsecdns_.com. At this point the root server will respond with the list of name servers that have authority over the com zone [21, 22], including A.GTLD-SERVERS.NET and its correct IP address (as a "glue" record [21, 22]). The RDNS will then compare this IP to the attacker's IP, notice the difference, and therefore discard the attacker's attempt to overwrite the cache entry for A.GTLD-SERVERS.NET. This verification process adds very little DNS traffic because of the fact that legitimate events that overwrite the IP address of TLD name servers in the RDNS's cache are very rare events, and also the number of different TLD name servers is small. Therefore, unless under poisoning attack, the RDNS will need to initiate a verification process only in rare cases. It is worth noting that given the presence of the <rand> string in the TLD verification query, poisoning a TLD name server now becomes practically infeasible (see Section 3.5). This means that WSEC-enabled zones will be protected from poisoning attacks along the entire resolution path without requiring any configuration change at the root and TLD name servers.

[6]The list of domain names and IP addresses for the root name servers are hardcoded into the RDNS software [21], and should be updated only by updating the RDNS software, its local configuration, or by very infrequent queries (e.g., once every several weeks) that cannot be forced/initiated by the attacker. Therefore, a poisoning attack involving the root name servers should always fail.

3.5 Robustness to Poisoning Attacks

Let Γ_x be the cardinality of the search space introduced by technique x. In the following we will assume $\Gamma_{TXID} = 2^{16}$ (perfect randomization of the TXID), and $\Gamma_{port} = 2^{16} - 1024$ (i.e., perfect randomization of all the ports excluding the first usually reserved 1024 ports). For the 0x20-bit encoding we consider three cases, namely $\Gamma_{0x20} = 2^6$ for short domain names containing 6 letters (e.g., cnn.com, aol.com, etc.), $\Gamma_{0x20} = 2^{12}$ because [8] reports 12 as the average length of domain names, and $\Gamma_{0x20} = 2^{16}$. Also, we assume WSEC DNS queries are performed using random alphanumeric prefixes of length 5, thus $\Gamma_{WSEC} = 36^5$ (WSEC random strings consists of combinations of lower-case letters and digits). Similar to previous work [14], in our analysis we take into account a round-trip-time (RTT) between the RDNS and an ANS equal to $\Delta T = 100ms$, and the number of available ANS for a given domain $\Gamma_{ANS} = 4$ (we found that "popular" domain names like cnn.com, aol.com, google.com, etc., use 4 ANS). For the attack scenario, we consider an attacker who is able to launch a brute-force DNS cache poisoning attack using Kaminsky's technique [15]. We assumed the forged responses are $l = 80$ bytes long (similar to [14]). Also, we consider three possible scenarios, in which the attacker has a total available network bandwidth BW of 1Mbps, 10Mbps and 100Mbps (higher attack BW may cause a denial of service on the RDNS, instead of poisoning the cache, in many practical scenarios). Furthermore, we assume the attacker launches an instance of the attack using its entire bandwidth to "flood" the RDNS with forged responses within the $\Delta T = 100ms$ time. If the first round is not successful, the attacker immediately retries with a new instance of the attack. Therefore, the attack frequency is $f_{attack} = \Delta T^{-1} = 1/100ms$ (i.e., the attacker repeats Kaminsky's attack every $100ms$)[7].

Suppose the attacker is able to send M spoofed DNS answers to the RDNS server within the RTT, after which the RDNS receives the answer from the real authoritative name server. The number M depends on the bandwidth BW available to the attacker, the RTT ΔT and the size l of the response packets. The probability of successful cache

[7]This is different from "traditional" poisoning attacks, in which case the attacker would have to wait for the entire TTL of the domain name to be poisoned (e.g., the TTL of the targeted domain name may be several minutes, or even hours). It is intuitive that Kaminsky's attack greatly increases the chances of successful attack within a given time window.

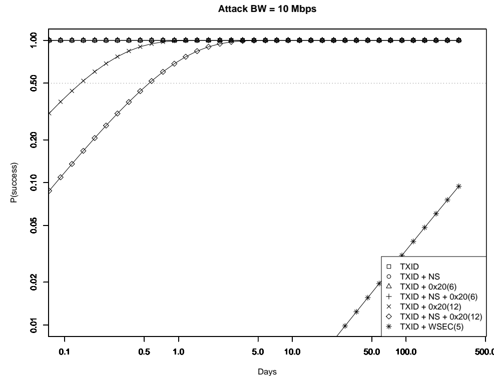

Figure 6: Effect of port *derandomization* due to NAT/PAT devices on the probability of success for Kaminsky's attack, and comparison with WSEC DNS. This case represents a scenario in which the RDNS's source port is made easily predictable (e.g., static, sequential, or round robin).

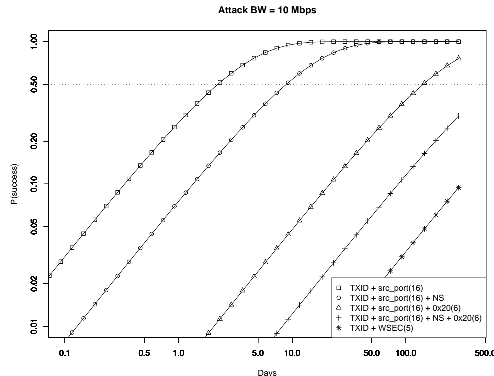

Figure 7: Effect of port randomization on probability of success for Kaminsky's attack, and comparison with WSEC DNS.

poisoning for one attack attempt can be computed as [12]

$$p_{succ} = 1 - p_{fail} = 1 - \prod_{i=0}^{M-1} \left(1 - \frac{1}{\Gamma - i}\right), \quad \Gamma > M - 1 \tag{1}$$

assuming the forged responses are generated without repetitions[8]. Equation (1) represents the probability of success of a single instance of the attack. Γ represents the overall cardinality of the search space. The total probability of success after launching n instances of the attack can be computed as $P_{succ} = 1 - (1 - p_{succ})^n$.

Figure 7 reports how the probability P_{succ} varies for growing attack durations (i.e., growing n), considering an attacker who has access to a bandwidth of $BW = 10Mbps$. Figure 7 considers the case in which port randomization can be effectively used, whereas Table 1 show the effect of port derandomization due to devices such as firewalls,

	BW = 1Mbps	BW = 10Mbps	BW = 100Mbps
TXID	29.08 seconds	2.87 seconds	0.25 seconds
TXID+0x20(6)	31.06 minutes	3.1 minutes	0.31 minutes
TXID+0x20(12)	33.13 hours	3.31 hours	0.33 hours
TXID+0x20(16)	22.09 days	2.21 days	0.22 days
TXID+ANS+0x20(6)	2.07 hours	0.21 hours	0.02 hours
TXID+ANS+0x20(12)	5.52 days	0.55 days	0.055 days
TXID+ANS+0x20(16)	88.35 days	8.82 days	0.88 days
TXID+WSEC(5)	55.83 years	5.58 years	0.56 years

Table 1: Effect of port *derandomization* due to NAT/PAT devices on the time needed to reach $P_{succ} = 0.5$ using Kaminsky's attack for different combinations of defense scenarios and attacker's bandwidth.

load-balancers, network address translators, etc., which perform port translation without preserving randomness [10] (we refere to such devices as NAT/PAT). The numbers between parenthesis in the legend of both Figure 7 and Table 1 represent the additional bits of entropy for the random source port and 0x20-bit encoding techniques, and the length of the random prefix strings for WSEC DNS queries.

It is clear from Table 1 that the TXID is not sufficient to protect against Kaminsky's attack, because in this case the expected time $\bar{T}_{attack}$ after which $P_{succ} = 0.5$ is in the order of seconds (the exact value depends on the attacker's bandwidth). Even combining the TXID with the random choice of an ANS and 0x20-bit encoding for domain names containing 6 letters, $\bar{T}_{attack}$ is in the order of minutes. In case of longer domain names, e.g., names containing 12 letters, 0x20-bit encoding in combination with the random ANS choice offers a little better protection, bringing $\bar{T}_{attack}$ up to the order of days, although a successful attack is still quite feasible for a motivated attacker (it would take about 13 hours to reach $P_{succ} = 0.5$ with $BW = 10Mbps$). 0x20-bit encoding starts providing better protection only when 16 letters or more are present in the domain name. On the other hand, regardless of the length of the domain name, WSEC DNS provides a robust solution to cache poisoning attacks, including Kaminsky's attack [15], because even in absence of effective source UDP port randomization an attacker would need to persistently attempt the poisoning attack for years before reaching a 50% probability of success. For example, in order for an attacker to reach $P_{succ} = 0.5$ with a bandwidth $BW = 10Mbps$, it would take a time $\bar{T}_{attack}$ of more than 5 years. By increasing the length of the random prefixes used in WSEC DNS queries, the time needed for a successful poisoning attack increases exponentially. For example, if we increased the length of the random prefix strings to 6, even with a bandwidth for the attacker equal to $BW = 1Gbps$ it would take a time $\bar{T}_{attack}$ of about 2 years to reach $P_{succ} = 0.5$.

4 Experiments

We developed a proof-of-concept implementation of WSEC DNS on top of the PowerDNS recursive DNS re-

[8] More precisely, Equation (1) should be $p_{succ} = 1 - \prod_{i=0}^{M-1} \left(1 - \frac{q}{\Gamma - i}\right)$ [12], but in the following we assume the RDNS implements defense mechanisms against *birthday attacks* [25] (like most RDNS software), and therefore we set $q = 1$.

DNS Server	# queries to ANSs	# SERVFAIL	Median RTT	DNS traffic	Max Cache
pdns	220,930	3,892	52 ms	28.67 MB	3.25 MB
pdns+0x20	229,157	4,093	73 ms	30.04 MB	3.25 MB
pdns+WSEC+0x20(16)	255,605	4,066	87 ms	37.22 MB	4.79 MB
pdns+WSEC	269,156	4,125	90 ms	41.11 MB	5.57 MB

Table 2: Comparison between PowerDNS, 0x20 and WSEC DNS.

solver version 3.1.7 (`www.powerdns.com`). Our implementation of WSEC DNS consists of about 250 lines of C code[9]. We experimented with PowerDNS (referred to as *pdns* in the following) with no modifications, 0x20-bit encoding implemented on top of pdns (pdns+0x20), and WSEC DNS implemented on top of pdns (pdns+WSEC). Also, we experimented with a combination of WSEC DNS and 0x20, which we refer to as pdns+WSEC+0x20(16). In this latter case, the WSEC DNS query process (see Section 3) is activated only when the queried domain name contains less than 16 letters. Otherwise, only the 0x20-bit encoding is used. We performed all the experiments running the four different versions of the DNS resolver software on Xen (`www.xen.org`) virtual machine images with the same characteristics, namely a Linux-based operating system and 512MB of RAM per virtual machine. We collected a dataset of real DNS queries towards the RDNS resolver of a large enterprise network. The collected dataset contained a total of 5,539,540 queries to 473,487 distinct domain names. These queries were issued by 24,140 distinct hosts in a period of one day. We then extracted a random sample of 100,000 queries from the collected dataset and replayed this set of DNS queries for each of our experiments with different configurations of the DNS resolvers. The sampled 100,000 queries where related to 40,081 distinct domain names. As we do not have control over the ANSs related to the domain names in the replayed queries, all the results presented below are related to non-WSEC-enabled zones, and therefore represent a measure of the price to pay in order to maintain backward compatibility. In order to estimate the latency introduced by WSEC DNS queries, for each query we measured the Round Trip Time (RTT) between the stub-resolver and the RDNS. Namely, given a query q issued by a host behind the RDNS, let t_1 be the time when q is sent to the RDNS resolver, and t_2 be the time when the response from the RDNS resolver was received. In this case $RTT(q) = t_2 - t_1$. We also measured the amount of DNS traffic flowing from the RDNS to authoritative name servers, and the amount of cache used by the RDNS in different scenarios.

The results of our experiments are summarized in Table 2. By replaying our random sample of 100,000 queries mentioned above, the total number of queries to external ANSs generated by the RDNS using the original implementation of pdns is 220,930. The additional 120,930 queries are due to the fact that each query from the stub-resolver to the RDNS causes the RDNS to initiate a recursive "discovery" process to find an ANS that is able to provide an authoritative response to the query. Among the 100,000 queries initiated by the stub-resolver, 3,892 returned a *Server Failure* message, which means that no valid ANS was found for the requested domain. On the other hand, using pdns+WSEC the RDNS generated 269,156 DNS queries to ANSs, and returned 4,125 *Server Failure* messages to the stub-resolver. The increase in query volume is due to the additional "handshake" queries. It is worth noting that the WSEC DNS query process is applied by the RDNS to the original query from the stub-resolver as well as to all the queries needed to resolve ANSs *out-of-bailiwick* (i.e., the ones for which the "glue" records in the additional section of the response cannot be trusted). The increase in queries from the RDNS to the ANSs is the major cause of increase in RTT. During all our tests we noticed no significant increase in CPU usage on the RDNS resolver. The number of additional queries depends on the "locality" of the queries from the stub-resolvers, i.e., the proportion between the number of distinct domains queried and the number of distinct *zones* in which they are "located". An extensive study of these factors and their impact on the effectiveness of the cache for WSEC DNS is quite complex and is deferred to future work. The slight increase in the number of *Server Failure* messages for pdns+WSEC is due to an artifact of our implementation. Every time a "handshake" `TXT` query from the RDNS to an unreachable (or malfunctioning) ANS causes a *Server Failure* message, our implementation of pdns+WSEC assumes the zone is not WSEC-enabled and retries to query by omitting the WSEC random prefix. However, since the ANS is unreachable, this second step is unnecessary, because it will return the same exact *Server Failure* message. This problem can be solved by optimizing the implementation of our WSEC DNS query algorithm. Considering that the RDNS needs to wait for 3 seconds before declaring an ANS unreachable and send the *Server Failure* message to the stub-resolver, optimizing our WSEC DNS implementation would also contribute to decrease the median RTT.

From Table 2, it is easy to see that the increase in security provided by WSEC DNS comes at a price in terms of latency (increase in RTT), DNS traffic, and memory usage. However, the overhead due to WSEC DNS can be in part mitigated by software optimization and by combining it with the 0x20-bit encoding (e.g., for domain names that

[9]Our implementation of WSEC DNS can be downloaded from `http://roberto.perdisci.com/projects/wsecdns`.

contain 16 letters or more). We argue that the price to be paid in terms of increase in overhead is not that high, in particular in comparison with pdns+0x20, if we consider the obtained improvement in security with respect to using only 0x20-bit encoding (see Section 3.5), and considering that complete backward compatibility is maintained.

5 Conclusion

We presented Wildcard-Secure DNS (WSEC DNS), a novel DNS query process that wisely combines the use of wildcard domain names and `TXT` resource records to protect recursive DNS resolvers (RDNS) from brute-force cache poisoning attacks. We showed that WSEC DNS makes cache poisoning attacks, including Kaminaksy's attack [15], practically infeasible, even for very motivated and powerful attackers. Also, WSEC DNS provides complete backward compatibility with name servers that do not intend to support WSEC DNS queries, thus allowing for an incremental deployment. Our approach does not require any changes at the root and top-level-domain (TLD) name servers, thus allowing WSEC DNS to be deployed on a large-scale in a short period of time. This is in contrast with DNSSEC, for which a large-scale deployment seems to be far in the future.

Acknowledgements

We would like to thank David Dagon and Robert Edmonds for their insight into the DNS protocol, Prof. Farnam Jahanian for his guidance with improving the paper, Scott Rose, Tom Karygiannis, and the anonymous reviewers for their helpful comments.

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] AR. DNS multiple race exploiter: DNS cache poisoner/overwriter, 2008. http://www.securebits.org/dnsmre.html.

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

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol modifications for the DNS security extensions, March 2005. http://www.ietf.org/rfc/rfc4035.txt.

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

[5] D. J. Bernstein. djb-dns. http://cr.yp.to/djbdns.html.

[6] D. J. Bernstein. DNSCurve. http://dnscurve.org.

[7] D. J. Bernstein. The DNS Security Mess, 2006. http://cr.yp.to/talks.html#2006.10.17.

[8] D. Dagon, M. Antonakakis, P. Vixie, J. Tatuya, and W. Lee. Increased DNS forgery resistance through 0x20-bit encoding. In *ACM ACM CCS*, 2008.

[9] D. Dagon, N. Provos, C. Lee, and W. Lee. Corrupted DNS resolution paths: The rise of a malicious resolution authority. In *NDSS*, 2008.

[10] C. R. Dougherty. Vulnerability note VU#800113, 2008. https://www.kb.cert.org/vuls/id/800113.

[11] D. Eastlake and C. Kaufman. Domain name system security extensions, January 1997. http://www.ietf.org/rfc/rfc2065.txt.

[12] A. Householder and I. A. Finlay. Vulnerability note VU#457875, 2004. https://www.kb.cert.org/vuls/id/457875.

[13] J. G. Høy. Anti DNS spoofing - extended query ID (XQID), April 2008. http://www.jhsoft.com/dns-xqid.htm.

[14] A. Hubert and R. van Mook. Measures for making dns more resilient against forged answers, July 2008. http://tools.ietf.org/html/draft-ietf-dnsext-forgery-resilience-06.

[15] D. Kaminsky. Black ops 2008 – its the end of the cache as we know it. Presented at BlackHat 2008, 2008.

[16] A. Klein. BIND 9 DNS cache poisoning, 2007.

[17] O. M. Kolkman. DNSSEC basics, risks and benefits, 2005. https://ripe.net/info/ncc/presentations/domain-pulse.pdf.

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

[19] D. Leonard and D. Loguinov. Turbo king: Framework for large-scale internet delay measurements. In *IEEE INFOCOM*, 2008.

[20] E. Lewis. The role of wildcards in the domain name system, July 2006. http://www.ietf.org/rfc/rfc4592.txt.

[21] P. Mockapetris. Domain names - concepts and facil-

ities, November 1987. http://www.ietf.org/rfc/rfc1034.txt.

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

[23] J. Oberheide. Hardening DNS with IP TTLs, 2008. http://jon.oberheide.org/blog/2008/08/10/hardening-dns-with-ip-ttls/.

[24] E. Osterweil, M. Ryan, D. Massey, and L. Zhang. Quantifying the operational status of the DNSSEC deployment, 2008. UCLA Technical Report http://irl.cs.ucla.edu/papers/080019.pdf.

[25] J. Stewart. DNS cache poisoning – the next generation. http://www.secureworks.com/research/articles/dns-cache-poisoning/, 2002.

[26] M. Wong and W. Schlitt. Sender policy framework (SPF) for authorizing use of domains in e-mail, version 1, April 2006. http://www.ietf.org/rfc/rfc4408.txt.

A Additional Related Work

In XQID [13], Høy proposes to prepend a random string of length between 32 and 71 to domain names. For example, when querying for www.example.com the RDNS will format the DNS query to contain something similar to xqid--q.p0yw5c4eq0c2hszu.www.example.com. Similarly to WSEC DNS, XQID makes poisoning attacks practically infeasible, because now the search space for the attacker is huge and the chance of guessing a valid response is negligible, even if the attacker persistently tried to poison the cache for years (see Section 3.5). However, XQID suffers from a number of problems. XQID requires software updates on both the RDNS resolvers and the authoritative name servers [13], including the root and TLD name servers. This would clearly hamper a large-scale deployment of XQID. On the other hand, WSEC DNS does not require any change at the root and TLD name servers, and it only requires configuration changes for the other name server that intend to enable WSEC DNS queries. Also, XQID effectively shortens the maximum length of domain names defined by the DNS protocol, because up to 71 characters have to be reserved for the extended ID. In addition, backward compatibility is provided by identifying what name servers do not support XQID, and then requering for the original domain name, without any caching system. This has the effect of doubling the amount of DNS traffic towards the name servers that do not support XQID. WSEC DNS is significantly different from XQID in several ways, and does not suffer from the problems that afflict XQID. Differently from XQID, WSEC DNS is based on wildcard domain names, does not require software changes for name servers (including root and TLD name servers), and allows us to use variable length random strings as domain name prefixes. Also we show in Section 3.5 that less than 6 alphanumeric characters are sufficient to make poisoning attacks practically infeasible. Furthermore, differently from XQID, WSEC DNS provides complete backward compatibility without significantly increasing the DNS traffic towards name servers that do not support it, thus allowing for an incremental deployment.

Other security researchers have proposed to increase the entropy in DNS queries by allowing the QDCOUNT field to be greater than one, for example. This would allow the RDNS to insert more than one entry in the question section of DNS queries. The first entry may be used for the "real" query for which we require an answer, whereas the additional entries may be used to increase the entropy of the query. If name servers were to preserve all these entries in the question section of the response, and answer correctly to only the first query in the question section, this technique may be used to increase the entropy of DNS queries making poisoning attacks harder. However, most name servers do not currently support queries for which the QDCOUNT is greater than one and consider them as malformed queries. Therefore, a large-scale software update on both RDNS resolvers and name servers would be required for this solution to work. Furthermore, it is not clear if an incremental deployment would be possible at all. Another way to increase the entropy of DNS queries may be to use the unused bits of the QClass field. However, this would also force a large-scale software update without support for an incremental deployment.

B WSEC TXT vs. SPF

TXT resource records were originally meant for storing descriptive text about a domain name, but are now widely used to also carry information related to the Sender Policy Framework (SPF) [26] to mitigate the spam emails phenomenon. We would like to emphasize the fact the wildcard TXT records used for WSEC do not interfere with either SPF or simple descriptive text. In order to explain why this is the case, let us consider the zone file reported in Figure 4 as a configuration example. Ignore for a moment the lines marked with "; WSEC", and assume they were not present. Querying for example.com TXT? would return "v=spf1 a mx -all". On the other hand, querying for www.example.com TXT? would return "This is our website". Now, consider the configuration lines masked with "; WSEC". It is easy to see that querying for example.com TXT? or www.example.com TXT? still returns the expected answer as before. It is also worth noting that a query to <rand>._

wsecdns_.example.com TXT? still correctly returns "v=spf1 a mx -all", whereas a query to <rand> ._wsecdns_.www.example.com TXT? correctly returns "This is our website". Therefore, the introduction of WSEC does not alter the original information, and enables the protection of the RDNS's cache (thanks to the <rand> prefix) at the same time.

There is one potential issue: what happens if someone queries for the TXT record of a non-existant domain while expecting to obtain an SPF-related string? For example, a mail server is testing to see if mail apparently coming from abc.example.com should be blocked or not, according to SPF rules. If we do not consider the lines marked by "; WSEC", the mail server should receive a NXDOMAIN response. On the other hand, if the "; WSEC" lines are present, the mail server will receive the string "|wsecdns=enabled|". However, this does not represent a problem, because RFC 4408 (which describes the SPF protocol) explicitly says

> "Records that do not begin with a version section of exactly 'v=spf1' are discarded".

Therefore, the mail server in our example will discard the string "|wsecdns=enabled|", thus reducing to the case of no valid SPF entry for abc.example.com.

```
$ORIGIN example.com.
@       IN      SOA     ns1.example.com.  hm.example.com. (
                        2001062502 ; serial
                        21600      ; refresh after 6 hours
                        3600       ; retry after 1 hour
                        604800     ; expire after 1 week
                        600 )      ; minimum TTL 5 minutes
        IN      NS      ns1.example.com.
        IN      NS      ns2.example.com.
        IN      TXT     "v=spf1 a mx -all" ; SPF
        IN      MX      10      mail.example.com.
        IN      A       10.0.2.1
;
ns1     IN      A       10.0.2.2
ns2     IN      A       10.0.2.3
mail    IN      A       10.0.2.4
;
*                IN      CNAME   example.com.
;
;
;; RRs added for enabling WSEC DNS are reported below
;
_test_._wsecdns_  86400  IN    TXT   "|wsecdns=enabled|" ; WSEC
;
*._wsecdns_          IN      CNAME   example.com. ; WSEC
*._wsecdns_.ns1      IN      CNAME   ns1  ; WSEC
*._wsecdns_.ns2      IN      CNAME   ns2  ; WSEC
*._wsecdns_.mail     IN      CNAME   mail ; WSEC
;
*._test_._wsecdns_      IN   CNAME   _test_._wsecdns_ ; WSEC
*._test_._wsecdns_.ns1  IN   CNAME   _test_._wsecdns_ ; WSEC
*._test_._wsecdns_.ns2  IN   CNAME   _test_._wsecdns_ ; WSEC
*._test_._wsecdns_.mail IN   CNAME   _test_._wsecdns_ ; WSEC
```

Figure 8: Example-2 of WSEC-compliant zone file.

Consider now the zone file reported in Figure 8, and again ignore the lines marked with "; WSEC" for a moment. As we can see, in this case a wildcard "* IN CNAME" resource record exists. In this case, a wildcard "* IN TXT" record cannot coexist (according to RFC 1034) with the wildcard CNAME. Now assume no information is present about the zone example.com in the RDNS's cache. Also, assume the first query from a stub-resolver regarding a domain in that zone is about a non-existant (without considering wildcards) domain X.example.com. Given that no information about the zone is present in the cache, the RDNS will first perform the WSEC "handshake" querying for <rand>._test_._wsecdns_.X.example.com TXT?. This query will match the wildcard "* IN CNAME" and then return the SPF string "v=spf1 a mx -all" and the RDNS cannot determine if WSEC is enabled or not. In this case the RDNS can simply disambiguate by performing a query directly to <rand>._test_._wsecdns_.example.com TXT?, which either return "|wsecdns=enabled|" if WSEC is enabled, or NXDOMAIN. In our experiments we found that only about 7% of the top 500 domain names (ranked according to Alexa (www.alexa.com)) have a configuration similar to the one reported in Figure 8, where a wildcard "* IN CNAME" resource record exists. Also, it is worth noting that this causes the RDNS to perform a second WSEC "handshake" attempt only if, by chance, the first time a stub-resolver issues a query for a domain in one of these zones that does not exist (if we do not consider the wildcard).

Algorithm 1 WSEC DNS Query Process (simplified algorithm)

function WSecQuery :
input D {the domain to be resolved (e.g., www.example.com)}
input T {the query type requested by the host (e.g., A?)}
input N {the length of random prefix strings (e.g., 5)}
output {a DNS answer}

$\alpha \leftarrow$ GenerateRandomString(N)
$q \leftarrow$ Query(α._test_._wsecdns_.D,TXT) {issues a TXT? DNS query of the domain name α._test_._wsecdns_.D (e.g., a1b2c._test_._wsecdns_.www.example.com TXT?)}
$r \leftarrow$ GetNextValidAnswerTo(q) {multiple answers may be received if poisoning is attempted. Only the first answer that matches the actual queried domain name in the question section is accepted}

$w \leftarrow$ false
if Status(r)!=NXDOMAIN **then**
  $w \leftarrow$ isWSECEnabled(r) {w is true is r contains "|wsecdns=enabled|"}
  if w == true **then**
    AddZoneToPositiveWSECCache(D) {adds the zone of D in the positive WSEC cache}
    $\alpha' \leftarrow$ GenerateRandomString(N)
    $q' \leftarrow$ Query(α'._wsecdns_.D,T) {e.g., q'=d3rjf._wsecdns_.www.example.com}
    $r' \leftarrow$ GetNextValidAnswerTo(q')
    $r'' \leftarrow$ NormalizeWSECResponse(r') {see Algorithm 2}
  end if
end if
if w == false **then**
  AddZoneToNegativeWSECCache(D) {adds the parent zone of D in the negative WSEC cache}
  $q'' \leftarrow$ Query(D,T) {queries the original domain name and type}
  $r'' \leftarrow$ GetNextValidAnswerTo(q'')
end if

return r''

Algorithm 2 WSEC Response Normalization Process

```
function NormalizeWSECResponse :
input r {a WSEC response}
output {normalized DNS response}

r′ ← CutWSECPrefixFromQuestionSection(r) {r′ is equal to r, but the domain
name reported in the question section of r′ is normalized (e.g., cuts d3rjf.
_wsecdns_ from d3rjf._wsecdns_.www.example.com}
if Status(r)==NXDOMAIN then
    return r′
else
    q ← ExtractQueriedDomainFromQuestionSection(r′) {e.g., q=www.
    example.com}
    l ← ExtractFirstEntryFromAnswerSection(r′)
    l′ ← CutWSECPrefixFromDomain(l) {e.g., cuts d3rjf._wsecdns_ from
    d3rjf._wsecdns_.www.example.com}
    if RRTypeOf(l)==CNAME AND l′ == q then
        {the condition l′ == q makes sure that the CNAME is due to a WSEC
        query and points to a name in the same zone}
        r′′ ← DeleteFirstEntryFromAnswerSection(r′)
    else
        r′′ ← RewriteFirstEntryOfAnswerSection(r′,l′) {this basically cuts the
        WSEC prefix from the domain name in the first entry of the answer section
        of the DNS response}
    end if
end if

return r′′
```

C WSEC DNS for Delegated Sub-Zones

In Section 3.3 we assumed that if, for example, the zone `example.com` is WSEC-enabled, all its sub-zones are also WSEC-enabled. This is recommended. However, given that one of the requirements of WSEC DNS is the support of incremental deployment, enabling WSEC for all the sub-zones at the same time may not be easy (in particular for large organizations). We can easily solve this problem by following the delegation information contained in DNS responses coming from the authoritative name servers (ANS) for `example.com`. As an example, consider the following scenario. Assume `subzone.example.com` is a subzone of `example.com`. Let `www.subzone.example.com A?` be the query issued by the stub-resolver, and assume the WSEC positive cache of the RDNS indicates that WSEC is enabled for the zone `example.com`, but no WSEC cache entry exists for `subzone.example.com`. At this point the RDNS will issue the query `<rand>._wsecdns_.www.subzone.example.com A?` and send it to the ANS for `example.com`, say `ns.example.com`. As `ns.example.com` has delegated the authority over `subzone.example.com` to a different ANS, say `ns.subzone.example.com`, the response from `ns.example.com` will contain delegation information (in the authority and additional sections of the response). This allows the RDNS to realize that there is a zone delegation, and because there is no information about the support of WSEC for `subzone.example.com` in the WSEC cache, the RDNS will need to query `<rand>._test_._wsecdns_.www.subzone.example.com TXT?`, first, in order to retrieve the correct WSEC prefix. Assuming the response indicates `subzone.example.com` is WSEC-enabled, the RDNS will now issue the query `<rand>._wsecdns_.www.subzone.example.com A?`. On the other hand, if the ANS `ns.subzone.example.com` does not support WSEC queries, the RDNS will issue the "standard" query `www.subzone.example.com A?` and store the zone `subzone.example.com` in the negative WSEC cache.

D Analysis of Open Recoursive DNS Resolvers and Port Randomization

In order to understand to what extent UDP source port randomization has been adopted so far, we studied the behavior of open-recoursive DNS resolvers (O-RDNS), i.e., RDNS resolvers that accept and respond to recoursive DNS queries received from anywhere on the Internet [9]. Our objective is to verify what is the percentage of O-RDNS that use port randomization, and how many O-RDNS relay on another network device that forwards DNS queries to authoritative name servers (ANS) on their behalf. To this end, we use a methodology similar to the one proposed in [9]. In practice, we send queries to the O-RDNS "forcing" it to contact an authoritative name server (ANS) under our control. This can be easily done by registering a number of domain names, and then quering an O-RDNS for the domains (and their sub-domains) that we registered.

We experimented with 261,630 distinct active O-RDNS resolvers. To each one of them, we sent DNS queries crafted so that the O-RDNS is "forced" to contact our ANS 40 times in a very short period of time[10]. We then measured how many distinct source UDP ports where used by each O-RDNS that contancts our ANS as a consequence of our "stimulus" DNS queries. This allowed us to verify if the O-RDNS was using a fixed port, variable port numbers in a guessable sequence, or a randomized port. Also, we were able to measure whether an O-RDNS relays on another network device that forwards its packets to the ANS. This can be done by simply comparing the IP address of the O-RDNS to which we sent the query, and the IP address from which the ANS receives the query we just sent. We prepend a unique identifier to each query in order uniquely identify each O-RDNS we tested and to exclude "spurious" queries received by our ANS.

We found that among the 261,630 O-RDNS we considered in our experiments, only 38,381 (14.61%) contacted our ANS directly, whereas 223,249 (85.39%) contacted our ANS through another device (i.e., our ANS received queries from a different IP address than the exepected O-RDNS's IP to which we sent the "stimulus" quereis). This result is in accordance with the findings in [9]. Of the 38,381 O-

[10] We use 10 distinct consecutive queries. Each query forces the RDNS to follow a CNAME chain of length 4.

RDNS resolvers that contacted our ANS directly, 25,386 (66.14%) did so using a fixed (non-random) source UDP port. On the other hand, among the 223,249 O-RDNS that contacted our ANS through another device, we found that 83,691 ORDNS (37.49%) of them relied on a device that forwards quries on their behalf to our ANS using a fixed source UDP port. Although from our experiments we cannot draw exact conclusions on the nature of the external devices that forward packets on behalf of O-RDNS resolvers (e.g., whether these devices are load-balancers, firewalls, NAT/PAT, etc.), we can certainly notice that port randomization is not yet largely adopted by neither many of such devices, nor by many of the O-RDNS that contacted our ANS directly. Therefore, an attacker may be able to launch a cache poisoning attack towards these O-RDNS resolvers, and successfully poison the cache in a very short time (minutes or even seconds).