Ghost Domain Names: Revoked Yet Still Resolvable

Jian Jiang, Jinjin Liang
Network Research Center
Tsinghua University
{jiang-j08, liangjj09}@mails.tsinghua.edu.cn

Kang Li
Department of Computer Science
University of Georgia
kangli@cs.uga.edu

Jun Li
University of Oregon
Carlos III University of Madrid
Institute IMDEA Networks
lijun@cs.uoregon.edu

Haixin Duan*
Network Research Center
Tsinghua University
duanhx@tsinghua.edu.cn

Jianping Wu
Network Research Center
Tsinghua University
jianping@cernet.edu.cn

Abstract

Attackers often use domain names for various malicious purposes such as phishing, botnet command and control, and malware propagation. An obvious strategy for preventing these activities is deleting the malicious domain from the upper level DNS servers. In this paper, we show that this is insufficient. We demonstrate a vulnerability affecting the large majority of popular DNS implementations which allows a malicious domain name to stay resolvable long after it has been removed from the upper level servers. Our experiments with 19,045 open DNS servers show that even one week after a domain name has been revoked and its TTL expired, more than 70% of the servers will still resolve it. Finally, we discuss several strategies to prevent this attack.

1. Introduction

The Domain Name System (DNS), which provides a global mapping service between Internet domain names and IP addresses, is one of the most important components of the Internet. While primarily used for legitimate purposes, domain names have also been heavily leveraged by malicious activities such as phishing, malware propagation, and botnet command and control. A major endeavour in stopping these malicious activities has thus been identifying and deleting malicious domain names. For example, recent domain name takedown efforts have successfully shut down large scale botnets such as Waledac and Rustock [7].

While these successes have demonstrated that domain name revocation is effective in fighting against malicious activities, in this paper we show that removing malicious domains from domain registry is not enough to revoke the domain and IP mapping at the global scale. In fact, an attacker can keep their domain names continuously resolvable to attacker-controlled IP addresses—even after the original delegation data has been removed from the domain registry and the original time-to-live (TTL) period has expired. Because of the elusive nature of these domain names (i.e. revoked but still resolvable), we call them *ghost domain names*.

Ghost domain names are results of a vulnerability in the DNS cache update policy which prevents effective domain name revocation. The normal process of revoking a domain name from the global DNS system includes two steps: first the removal of the delegation data at the domain registry, and second the removal of all the cached copies throughout DNS resolvers. The first step is based on an explicit action, and the second one is implicitly governed by the TTL value associated with the delegation data. Although the TTL-based implicit revocation mechanism is not timely, it is still acceptable if a revoked domain name can be eventually cleared from every DNS resolver after the specified time according to the TTL value. Unfortunately, DNS allows a cached entry to be overwritten at a DNS resolver and the cache update logic is not strictly defined. An attacker can manipulate the cached delegation data and extend the TTL value. The attacker only needs to generate a recursive query to be resolved by the authoritative server (controlled by the attacker) before the TTL expires, and piggyback new delegation data in the crafted response from the authoritative server to the resolver (the victim). The new delegation data still resolves to an IP address controlled by the attacker but with new TTL value, and the attacker can continuously keep the delegation data alive in the resolver by repeating

*Corresponding author

the same attack.

This vulnerability is different from the notorious cache poisoning attacks [4]. While cache poisoning attacks compromise the integrity of DNS data by forging DNS responses, to exploit the ghost domain vulnerability, the attacker only needs to perform two legitimate actions: 1) querying the victim DNS resolver for a ghost domain to force the victim resolver to query the attacker's authoritative server before the delegation data expires, and 2) piggybacking new delegation data in the response from the authoritative server to the victim resolver. Because the two actions are seemingly legitimate, the vulnerability has not been addressed by the previous patches for cache poisoning.

Our study confirms that, until the writing of this paper, the majority of public DNS servers and up-to-date versions of popular DNS implementations, including the leading vendor BIND, are vulnerable. We believe that this vulnerability has not been suitably acknowledged by the networking and security community. Through experiments with 19,045 open DNS resolvers, we demonstrate that over 93% of experimental DNS resolvers are vulnerable and a large scale exploitation is practical. we have successfully created and kept ghost domain names in over 70% of experimental resolvers after one week of domain name revocation and TTL expiration.

This paper also discusses various approaches to addressing the problem of ghost domain names. Our study finds three DNS implementations, namely Unbound, MaraDNS and Microsoft DNS, have non-vulnerable versions. By comparing them, we find that although Unbound's strategy is simple in terms of implementation, MaraDNS's strategy is more fundamental — tightening the policy for delegation data update. We recommend that the DNS community adopt this defense strategy, and moreover consider a rigorous definition of DNS cache update policy.

The rest of this paper is organized as follows. We review necessary background of DNS in section 2. We then describe technical details of the vulnerability in section 3. In section 4 we present our exploitation experiments and analyze experimental results. We then discuss defenses and possible issues in section 5. Finally, we highlight the related work in section 6, and conclude the paper in section 7.

2. Background

In this section, we briefly summarize how DNS works. We focus on the concepts and details related to the vulnerability that we will present in the next section. Please refer to the DNS specifications for detailed descriptions [20] [21].

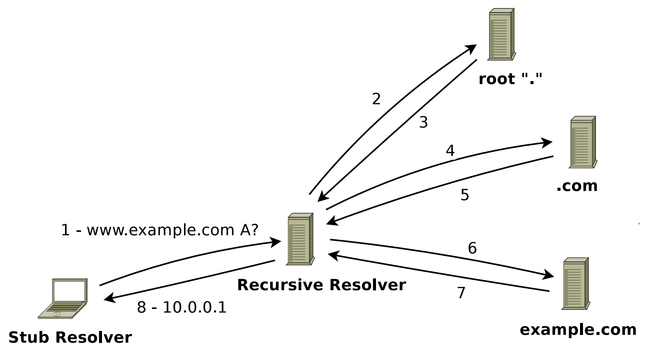


Figure 1. An example of DNS resolution process.

2.1. DNS Overview

DNS is organized around a hierarchical tree structure. Each domain name is composed of labels separated by dots. The domain name tree is divided into a series of *zones* based on the individual labels. Each zone represents a domain, and the server which holds the DNS data for all names under it is called the zone's *authoritative server*. An authoritative server may also delegate the authority of its sub-domains to other servers, which then serve as authoritative servers respectively for the sub-domains.

To a DNS client (*stub resolver*), a typical DNS name resolution process involves the client's local *recursive resolver* going through a series of queries to the authoritative servers of sub-domains along the tree of DNS. For example, Figure 1 shows a DNS resolution process of `www.example.com` through the three corresponding authoritative servers for the root zone, its child `.com`, and the next level sub-domain `example.com` respectively. Briefly, a stub resolver first requests a recursive resolver to resolve the domain `www.example.com`. Assuming the recursive resolver has no previous information about the domain, it will contact external servers in an iterative way. This iterative process includes a query to a root server (step 2) which redirects the recursive resolver to the `.com` authoritative server (step 3). Then the recursive resolver contacts the `.com` server (step 4), and from the reply it gets the information of authoritative server of `example.com` (step 5). The recursive resolver queries the authoritative server of `example.com` (step 6) which responds the IP addresses (step 7). The recursive resolver at the end forwards the response back to the stub resolver (step 8). During the process, the recursive resolver also caches received DNS data for further resolutions.

2.2. DNS Data Structure

Since the vulnerability we will present involves carefully crafted DNS responses, we also provide a brief summary about the DNS resource record and the DNS response format as background.

DNS data is stored using a basic data structure called *Resource Record* (RR). Every RR record is a five-tuple $< name, class, type, TTL, data >$, where $< name, class, type >$ serves as the key of *data*, *TTL* is a time-to-live value in seconds that limits the lifespan of cached copies. There are many *types* of RR records. Specifically, an `A` record gives the IP address of the name, and an `NS` record is another name that indicates the name of the server which has been delegated to serve as the domain's authoritative server. An `NS` record together with a corresponding `A` record are also known as *delegation data*. Delegation data points to the authoritative server of a sub-domain and provides its IP address as well.

When a DNS server receives a query that requests one or more types of RR records of a given name, it replies with a response that consists of three sections[1]: answer section, authority section and additional section. When the current DNS server cannot directly resolve the name in question, the authority section and the additional section, also known as referral sections, are used to carry an `NS` record and a corresponding `A` record. These records provide delegation data of a sub-domain that is closer to the name in question. The message below is a sample response of Step 5 in Figure 1.

```
;; ANSWER SECTION

;; AUTHORITY SECTION
example.com.    86400   IN  NS  ns.example.com.

;; ADDITIONAL SECTION
ns.example.com. 86400   IN  A   10.0.0.1
```

After a DNS resolver receives a DNS response, it then can cache the resource records contained in the response. Before the resource records expire based on the associated TTL value, the resolver does not need to go through the steps illustrated in Figure 1 to obtain the same records. For the sample response above, for example, the resolver can cache the delegation data of `example.com` for 86,400 seconds.

2.3. DNS Cache Update Policy

DNS cache is critical to its scalability and performance, and it can significantly reduce the overhead of authoritative servers and the response latency. However, it also poses many security threats. In the infamous *DNS cache poisoning* attack [4], an attacker can inject bogus RR records into DNS resolvers to redirect users to malicious addresses. For a number of reasons, DNS is inherently vulnerable to this type of attacks. For example, the connectionless nature of DNS protocol makes it vulnerable to spoofing attacks. An attacker can also use a compromised or malicious authoritative server to piggyback bogus records in referral sections of a DNS response.

The research and DNS communities have adopted several techniques to harden DNS against cache poisoning attacks. These techniques can be categorized into two classes: techniques for increasing forgery resistance and techniques for tightening cache update policy. Here we focus on the second class of techniques, specifically, the *bailiwick rule* [31] and the *credibility rule* [13]. The bailiwick rule checks referral sections in a DNS response to see if its contained RR records are in the authority range of the asked authoritative server. For example, if a DNS response comes from authoritative server of `.com` and contains a `NS` record of `.net`, then this record is considered as "out-of-bailiwick" and should be discarded. The credibility rule gives each RR record a different trust level according to where the response comes from and in which section the record is contained. Only when a new RR record has a higher or equal trust level should a cached RR record be overwritten.

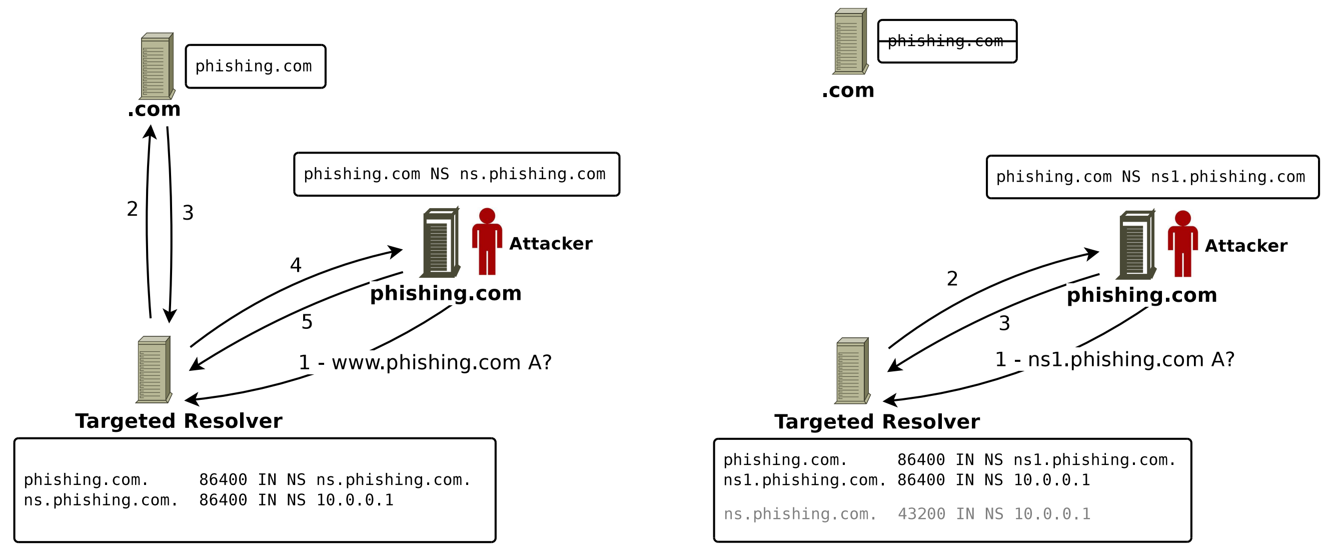
Unfortunately, both the bailiwick rule and the credibility rule have limitations. The bailiwick rule does not have a standard specification and depends on implementation. While the credibility rule specifies when cached entries can be overwritten, it still could be exploited for malicious purposes. For example, Dan Kaminsky reported a novel cache poisoning attack [18] that leverages non-existent names to increase spoofing efficiency, and in particular, exploits the credibility rule to overwrite cached entries by piggybacked referral sections. Some DNS vendors have responded to this attack by enhancing the validation of any RR records included in the additional section. However, in the next section, we demonstrate that even with such validation, the DNS cache update policy is still insecure.

3. The DNS Name Revocation Vulnerability

We illustrate the DNS name revocation vulnerability in this section, focusing on how an attacker can generate and maintain a ghost domain name.

To completely revoke a domain name (e.g., `phishing.com`), the delegation data for the name must be deleted from the authoritative servers of the parent zone of the name (e.g., `.com`), and sufficient time must be allowed for every recursive resolver to remove the cached delegation data for the name when the data expires. However, the current bailiwick and credibility rules that govern the overwriting of cached DNS resource records do not prevent an attacker from illegally renewing

[1] A DNS response also contains a question section copied from the original request, which we ignored for simplicity.



```
phishing.com.     86400 IN NS ns.phishing.com.
ns.phishing.com.  86400 IN NS 10.0.0.1
```

```
phishing.com.     86400 IN NS ns1.phishing.com.
ns1.phishing.com. 86400 IN NS 10.0.0.1

ns.phishing.com.  43200 IN NS 10.0.0.1
```

(a) Before `phishing.com` is deleted, the attacker pulls the delegation data for `phishing.com` into the victim DNS resolver.

(b) After `phishing.com` is deleted, the attacker manipulates the victim resolver to keep the delegation data in its cache.

Figure 2. An example ghost domain name (`phishing.com`) attack.

cached delegation data, even after the data has been deleted from the parent zone. An attacker can extend the TTL (time-to-live) value of the cached delegation data, therefore keeping a malicious domain (such as `phishing.com`) continuously resolvable.

Figure 2 shows a typical ghost domain name scenario, where an attacker manages to keep the delegation data of `phishing.com` in a victim recursive resolver. We assume that the attacker has registered a domain name `phishing.com` at the authoritative server of `.com`. Although this server is out of the attacker's control, the attacker runs and controls the authoritative server of `phishing.com`.

Two phases of establishing a ghost domain name are involved:

- **Phase 1: Caching the delegation data of a domain name.** The attacker targets a DNS resolver and requests it to resolve a name under `phishing.com`, say `www.phishing.com`. During the resolution process, the authoritative server of `.com` provides the victim resolver with the delegation data of `phishing.com`, such as:

```
;; ANSWER SECTION

;; AUTHORITY SECTION
phishing.com.     86400   IN  NS  ns.phishing.com.

;; ADDITIONAL SECTION
ns.phishing.com. 86400    IN  A   10.0.0.1
```

  The victim resolver accepts and caches the delegation data above. After $43,200$ seconds, as shown in Figure 2b, `phishing.com` is identified as a malicious domain and deleted from `.com`. At this moment, however, the victim resolver can still resolve `phishing.com` since it still caches the delegation data of `phishing.com`, which will not expire until another $43,200$ seconds later.

- **Phase 2: Refreshing the cached delegation data of the ghost name.** At some point after `phishing.com` has been removed from `.com` but before the delegation data of `phishing.com` expires, the attacker manipulates the victim resolver to conduct a series of DNS operations in order to have it continue to cache the delegation data after the original expiration time. These operations follow the standard DNS protocol without violating any rules, and are as follows.

  The attacker first changes the `NS` record of `phishing.com` to a new name, say `ns1.phishing.com`, then queries the victim resolver for the `A` record of `ns1.phishing.com`. Based on the cached, non-expired delegation data of `phishing.com`, the victim resolver learns and contacts the authoritative server of `phishing.com`, and receives a response, such as:

```
;; ANSWER SECTION
ns1.phishing.com. 86400   IN  A   10.0.0.1

;; AUTHORITY SECTION
phishing.com.     86400   IN  NS  ns1.phishing.com.

;; ADDITIONAL SECTION
```

```
ns1.phishing.com. 86400   IN  A   10.0.0.1
```

Both the answer section and the authority section conform to the bailiwick rule. Also, according to the credibility rule, the new `NS` record in the authority section has the same trust level as the old `NS` record of `phishing.com` in the cache. The victim resolver will therefore overwrite the old `NS` record with the new one, and the `A` record in the answer section and the new NS record will form the complete delegation data of `phishing.com`, i.e.:

```
phishing.com.     86400   IN  NS ns1.phishing.com.
ns1.phishing.com. 86400   IN  A  10.0.0.1
```

Most importantly, the new delegation data has a fresh TTL value of $86,400$, meaning the lifetime of `phishing.com` in the victim resolver starts over now for a new round of $86,400$ seconds. In fact, the attacker can refresh the delegation data later again and again before the data expires, thus making `phishing.com` accessible from the victim resolver for a very long time. In another words, the resolver will be continuously haunted by this ghost domain name, so we also call this resolver a *haunted resolver*.

The attacker can target many other DNS resolvers and repeat the same cache-and-refresh manipulation operations as described above with these victims. As a result, the attacker can use `phishing.com` to host their malicious sites for a long time, and users throughout the Internet would continue to be able to resolve `phishing.com` to attacker-controlled IP addresses.

The success of the ghost domain name attack assumes the attacker is able to send regular DNS queries to DNS resolvers. This assumption is practical when the attacks target open resolvers; previous research [11] has shown that there are still a large number of open resolvers around the world. The assumption is also practical if the attacker is in the service range of a DNS resolver, or he can control a bot machine in that range thus can initiate DNS queries from the bot machine.

4. Experiments, Results, and Analysis

While theoretically feasible according to our discussion in Section 3, it is unclear if an attacker can indeed launch a large-scale exploitation in the real world based on the ghost domain vulnerability of DNS, and if so, to what extent and at what cost. We investigate this matter in this section, focusing on the following questions:

- How many deployed DNS resolvers and DNS implementations are vulnerable?
- What would be the cost for an attacker to maintain ghost domain names?
- For those vulnerable DNS resolvers, how long can an attacker keep their ghost domain names in these resolvers?

4.1. Experimental Setup

We collected 19,045 open DNS resolvers from the query log of a busy authoritative server, also with the help of the authors of [28]. Table 1 shows how these resolvers are distributed around different geographic regions and different autonomous systems (AS).

Region	Count	Percentage
Japan	2479	13.01
USA	2471	12.97
Russian	1987	10.43
China	1742	9.15
Taiwan	1093	5.74
Germany	1020	5.36
Poland	547	2.87
Britain	546	2.87
Italy	512	2.69
HK	348	1.93
Total 161 regions		

(a) Regions

AS number	Count	Percentage
3462	628	3.29
538	455	2.52
4713	384	2.38
4134	351	2.01
1659	261	1.84
4837	257	1.37
4732	200	1.34
17506	164	1.05
9600	115	0.86
2907	106	0.60
Total 5474 ASes		

(b) Autonomous Systems (ASes)

Table 1. Statistics of DNS resolvers used in our experiments.

We conducted several experiments using these resolvers. We registered a domain name `ghostdomain.info`, and created ten sub-domains (`[1-10].ghostdomain.info`). We then conducted the Phase-1 operations (as described in Section 3) with every resolver we collected, so that they all have a cached entry for every sub-domain of `ghostdomain.info`. After four hours, we simultaneously remove all sub-domains from the authoritative server of `ghostdomain.info`, except for one of the ten sub-domains, `1.ghostdomain.info`. We then periodically conduct the Phase-2 operations (as described in Section 3) for *one* week on every resolver for the rest nine sub-domains, i.e., `[2-10].ghostdomain.info`. Each of these nine sub-domains has a different parameter setting for their domain names with regard to the original TTL value (1800, 3600, and 14400 seconds)[2] and the refreshing interval for Phase-2 operations (TTL/2, TTL/4, and TTL/8).

During the entire week, every ten minutes we probe every resolver to see if the cached entry of a sub-domain is still alive at every resolver, in order to learn the normal behavior after a domain name is revoked by testing `1.ghostdomain.info`, and how long a ghost domain name (`[2-10].ghostdomain.info`) can survive in every resolver. Every ten minutes we also check if every resolver can resolve `www.google.com` to make sure every resolver is still reachable and functioning.

4.2. Vulnerable Public DNS Servers and Popular DNS Implementations

Using `ghostdomain.info` under our control, we tested how public DNS servers and popular DNS implementations may be susceptible to the ghost domain name vulnerability. We discover that the distribution of the vulnerability is wide.

First, as shown in Table 2, we tested ten well-known public DNS servers from five service providers. Among tested servers, only the two from Google were not vulnerable. Since Google's DNS implementation was not open source, we could not know exactly how it avoided this vulnerability.

As shown in Table 3, we also chose six popular DNS vendors based on a recent DNS survey [30] and tested if they were vulnerable: *BIND, DJB dnscache, Unbound, PowerDNS, MaraDNS* and *Microsoft DNS*. For each vendor, we first tested its latest version. For those which were not vulnerable, we then tested their previous versions to see at which version the vulnerability got addressed. Five out of nine implementations, including the latest version from the leading vendor BIND, were vulnerable. We also found that three vendors had non-vulnerable versions, and we will discuss their defense strategies in Section 5.

[2]An attacker can set a very large TTL value for the delegation data of a malicious domain so that they can stay in cache for a very long time, even without launching the ghost domain name attack. But most DNS implementations have a maximum TTL limitation; they might force a DNS resolver to reset the TTL value of a cached entry, or even simply discard the entry if the original TTL exceeds the maximum TTL value allowed. We have made sure the TTL values we used here are less than the maximum TTL value allowed in DNS resolvers that we experiment with.

Service Provider	IP Address	Vulnerable?
Google	8.8.8.8	No
	8.8.4.4	No
DNS Advantage	156.154.70.1	Yes
	156.154.71.1	Yes
OpenDNS	208.67.222.222	Yes
	208.67.220.220	Yes
Norton	198.153.192.1	Yes
	198.153.194.1	Yes
GTEI DNS	4.2.2.1	Yes
	4.2.2.2	Yes

Table 2. Vulnerability testing of public DNS servers.

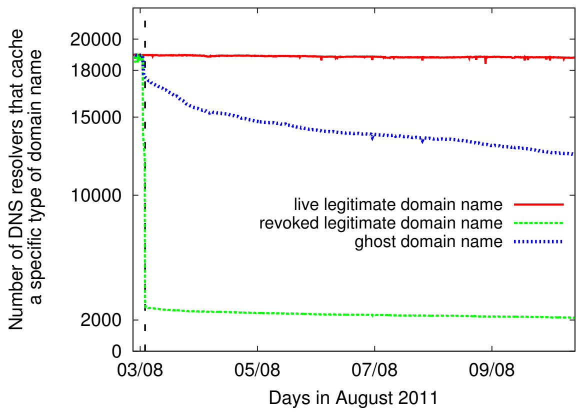
DNS Vendor	Version	Vulnerable?
BIND	9.8.0-P4	Yes
DJB dnscache	1.05	Yes
Unbound	1.4.11	No
	1.4.7	Yes
PowerDNS	Recursor 3.3	Yes
MaraDNS	Deadwood-3.0.03	No
	Deadwood-2.3.05	No
Microsoft DNS	Windows Server 2008 R2	No
	Windows Server 2008	Yes

Table 3. Vulnerability testing of popular DNS implementations.

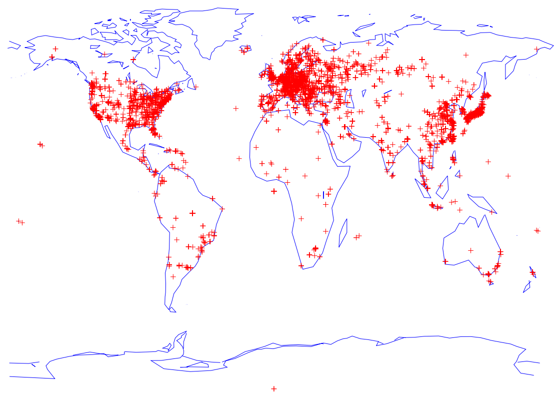
4.3. Efficacy of Maintaining Ghost Domain Names

We measured how the 19,045 open DNS resolvers may continue to resolve a ghost domain name (`2.ghostdomain.info`). For comparison, we also measured (1) how every resolver may continue to resolve a continuously existent, legitimate domain name, for which we use `www.google.com`; and (2) how every resolver may continue to resolve a legitimate domain name that gets revoked without the ghost domain operations; we use `1.ghostdomain.info` for this purpose. We therefore have three types of domain names: ghost domain name, live legitimate domain name, and revoked legitimate domain name.

Figure 3a shows how the DNS resolvers in our experiments may continue to resolve the three different types of domain names. By probing a resolver for a particular type of domain name, we can identify whether it is resolvable or not. From Figure 3a, it is clear that the three different types of domain names present different behaviors. As expected, a live legitimate domain name (`www.google.com`) can be resolved by almost all resolvers continuously. The num-



(a) Resolving of domain names at open DNS resolvers over time. The dashed vertical line indicates the original TTL expiration time for `[1-2].ghostdomain.info` after they are removed from domain registry.



(b) Geographic view of open DNS resolvers that are still haunted by ghost domain names one week later.

Figure 3. Measurement of ghost domain names at open DNS resolvers.

ber of resolvers that can resolve a revoked legitimate domain name (`1.ghostdomain.info`), however, rapidly falls down after its TTL expires. In contrast, a ghost domain name (`2.ghostdomain.info`) can be resolved by most of DNS resolvers as time goes by, even after its TTL expiration time. More than 93% of DNS resolvers still reply with a positive response for resolving the ghost domain name after its TTL has expired, meaning all these resolvers are vulnerable to the ghost domain name attack. Even one week after TTL expiration, the ghost domain names are still resolvable by more than 70% of all DNS resolvers we collected. Figure 3b provides a geographic view of all the haunted DNS resolvers at this time point. (We discuss later why the number of DNS resolvers that resolve a ghost domain name, i.e. the number of haunted DNS resolvers, gradually declines as time goes by.)

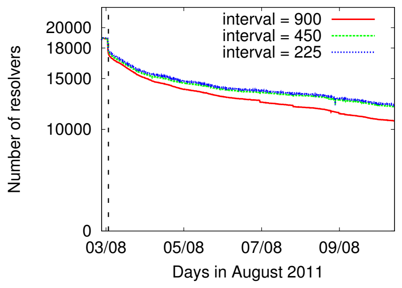
We note that after TTL expiration, a revoked legitimate domain name can still be resolved by approximately 10% of DNS resolvers, rather than 0% as we might expect. This phenomenon is due to extremely loose enforcement of cache update policy at certain resolvers that still employ very old versions of DNS implementations. They accept all referral sections and overwrite cached entries without any validation. These resolvers can be haunted by a ghost domain name even without the sophisticated attack as described in Section 3, but simply with periodical queries of a ghost domain name.

4.4. Cost of Maintaining Ghost Domain Names

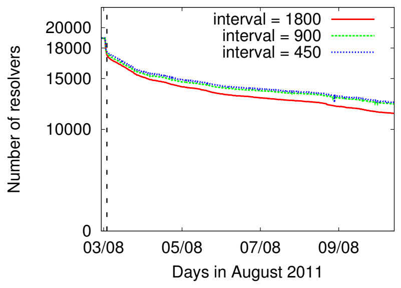
As we have shown the vulnerability of ghost domain in popular DNS implementations and widely deployed Internet DNS resolvers, a natural question is about the cost of launching and maintaining ghost domain names. Since the actions required by ghost domain attacks are sending queries to targeted resolvers, the main cost of maintaining a ghost domain name is network bandwidth consumption. An attacker needs to send plenty of DNS queries to refresh the cached entries of a ghost domain name, and the amount of such effort is proportional to the number of the targeted resolvers. For each individual targeted resolver, the bandwidth consumption of the ghost domain attack is mainly determined by the refreshing interval, which is related to two factors: the TTL value of the delegation data for the ghost domain and the refreshing interval per TTL. The refreshing interval relates to the TTL value because to keep the cached data alive, at least one refreshing operation is needed before a cached entry expires.

A naïve attacker might choose a very large TTL value of the delegation data with the hope of maintaining the ghost domain in a resolver's cache before refreshing it. This strategy saves bandwidth, but it is not practical since most of DNS resolvers enforce a maximum TTL limitation. We found empirically that most of our experimental resolvers have a maximum TTL limitation around one day (16.35%) or one week (79.81%), which means the TTL value of the ghost domain is limited.

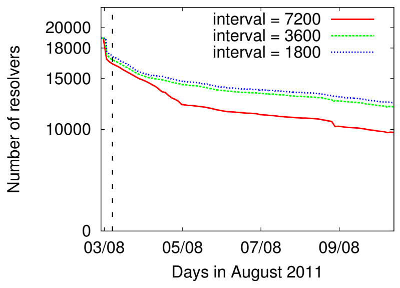
For any given TTL value selected by the attacker, the bandwidth consumption is directly related to how often attackers refreshes in one TTL. Figure 4 shows the effects of ghost domain exploitation with different TTL values and refreshing intervals. From the results, we observe that if the attacker ensures that one refreshment is sent for each quarter of TTL, additional refreshing effort can increase the effec-



(a) TTL = 1800



(b) TTL = 3600



(c) TTL = 14400

Figure 4. Establishment of ghost domain names with different TTL values and refreshing intervals.

tiveness, but not significantly. This is reasonable because the attacker only needs to ensure one successful renewal before the ghost domain expires. refreshing more often is useful only to reduce the chance of packet losses. We also see that the curves of $TTL/2$ refreshing interval declines faster, since only one refresh within the TTL period might occasionally fail, so a timely renewal of the ghost domain cannot be ensured.

4.5. Other Factors that Affect The Lifetime of Ghost Domains

Through all our experiment results such as those shown in Figure 4 and Figure 3a, we observe that the numbers of exploited resolvers continuously decrease over time. Although this declining trend is affected by the attackers' actions (such as the refreshing frequency discussed above), the types of the recursive resolvers turn out to affect the lifetime of ghost domain as well.

To illustrate the impact of recursive resolver types, we analyze the experimental data of probing `www.google.com` to study when and how frequently a cached entry (in this case, the `CNAME` record of `www.google.com` as it has a large original TTL) gets evicted. Knowing how a cache record could be evicted in a resolver helps us to infer why a ghost domain could be lost. By tracking the TTL variations of the objective entry, we find that the experimental resolvers can be sorted into four types based on the different TTL variations of the objective entry.

- *Stable resolvers* (Figure 5a). As expected, the TTL value monotonically decreases to zero, then go back to the original value.
- *Resolvers that occasionally become unreachable* (Figure 5b). Unreachable periods derive the gap in the middle of the line. Ghost domains could fail to be renewed during such periods.
- *Proxies* (Figure 5c). The TTL values form multiple lines, which means there are multiple cache servers in the backend. Ghost domains are easy to be lost after churn of backend servers.
- *Resolvers that occasionally lose cached entries* (Figure 5d). This may be caused by several factors: cache replacement, cache flushing, or even reset of resolvers. These factors also could cause the loss of ghost domains.

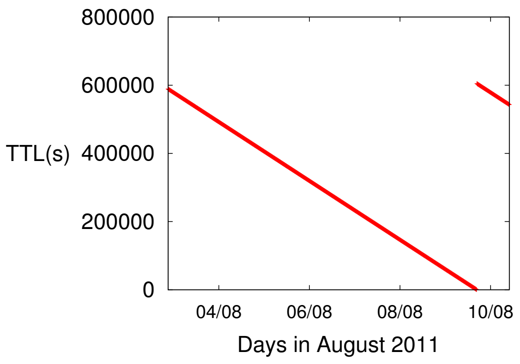
In our experiments, we find that over 85% of resolvers that fail to keep ghost domains belong to the last two types, which are inherently easy to lose cached entries. Less than 10% of failed resolvers come from the set of stable resolvers, which means stable resolvers are less likely lose cached entries, including ghost domains. Since over 65% of our experimental resolvers are stable from current observation of the TTL variations, we infer that a ghost domain can stay alive for a long time. In fact, after one week, over 70% of experimental resolvers still keep the ghost domains we created and the decline trend is slow. We are currently exploring methods to identify the types of resolvers and study their individual reaction to different ghost domain exploitations.

5. Discussion

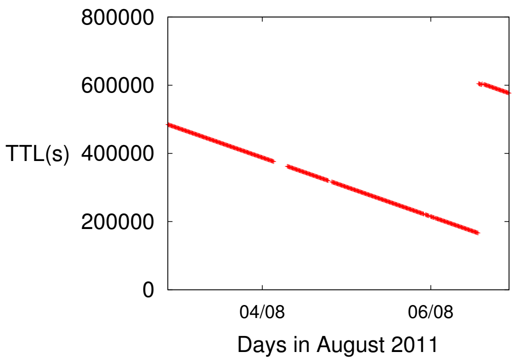
In this section, we discuss a few possible solutions to fix the problem of ghost domain names. We also discuss the current practices in some of the implementations that are not vulnerable to this problem. Finally, we conclude the section with a discussion of DNSSEC [2], which avoids this problem implicitly through its strict delegation requirements.

5.1. Defense Approaches

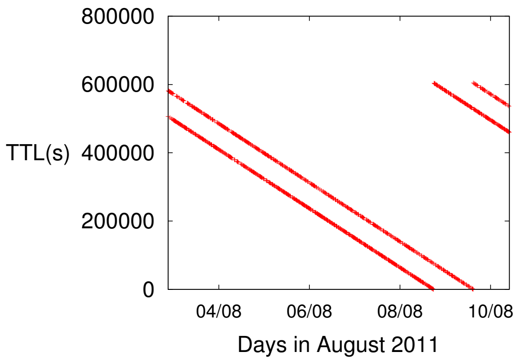
The ghost domain exploitation needs to launch a query to the target resolver. Thus a basic defense strategy is to



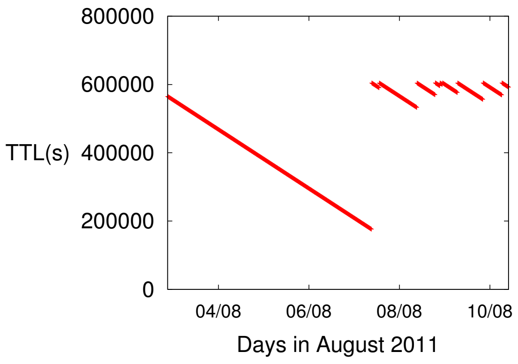
(a) Stable resolver



(b) Resolver with timeout



(c) Proxy



(d) Unstable resolver

Figure 5. TTL variations in different types of resolvers

have DNS administrators restrict the service range of the DNS resolver. Also, administrators can routinely flush DNS cache to purge possible ghost domains.

These strategies however are not fundamental solutions. The root cause of the ghost domain problem is that the current DNS cache update policy allows authoritative servers to continuously renew their own delegation data in resolvers by themselves. Then once a domain delegates a sub-domain, there is no guarantee that the delegation can be revoked.

Intuitively, there are several approaches for correcting this problem, and we consider the following three:

1. *Strengthening the bailiwick rule* – DNS resolver implementation should tighten the bailiwick rule so that a recursive resolver only accepts a zone's delegation data from authoritative server of its parent zone.

2. *Refining the credibility rule* – Another possible solution is to refine credibility to disallow cache overwriting when received records have the same trust level as cached data.

3. *Allowing updates with the exception of the TTL value* – Since the ghost domain attack achieves the goal of preserving revoked domain by refreshing the delegation data with a new authoritative server name and thus a new TTL, one possible solution is to allow the cache update with exception of the TTL value.

The first solution derives from the semantics of the bailiwick rule. The purpose of bailiwick rule is to restrict authoritative servers so that they can only give records in their own range. From this point of view, an authoritative server has no right to change delegation data of itself since the delegation should be dominated by its parent zone.

However, applying the strict bailiwick rule might cause performance and management issues. One such issue is resilience to authority mismatches. Authority mismatch is a type of DNS misconfiguration in which the delegation data is different in the parent zone than in the child zone. Although the DNS specification [20] requires that delegation data must be consistent, previous studies [30] [27] show this configuration error is common in practice. While the current cache update policy is resilient to such error, the strict bailiwick rule will ignore delegation data from the child zone, and thus might make some of authoritative servers unusable.

Another issue is about authoritative server migration. Allowing cached `NS` records to be overwritten can speed up legitimate migration of an authoritative server. However, with the strict bailiwick rule and current DNS protocol, resolvers will not be aware of the migration until cached delegation data expires. What's even worse, DNS administrators tend to give large TTL values to delegation data. We measured the TTL values of the delegation data of the top one million Internet domains (ranked by `alexa.com`), and we found that the TTL values of most popular domains are one (12.04%) or two days (78.41%). This study indicates that with the strict bailiwick rule, legitimate authority changes would take days to complete.

The second approach can thwart the ghost domain problem since all self-issued delegation data has the same trust level [13]. Also, this approach remains resilient to the authority mismatch problem, as the self-issued delegation data from the child zone has higher trust levels than those from the parent. This approach has the additional benefit of eliminating some of the attack vectors that could be exploited by cache poisoning attacks [31]. But it still suffers from the authority migration problem.

The third approach (limiting TTL updates) does not have penalties for the legitimate changes of authoritative servers. This approach also remains resilient to authority mismatch, and it is the simplest one in terms of implementation. However, we only consider this approach as a temporary solution — it does not actively address the issue of the loosely defined update policy.

Although the second approach has more advantages in practice, we prefer the first one as the recommended solution. A strict bailiwick rule that rejects self-issued delegation data is semantically correct. More importantly, the DNS standard must clarify and formally define the cache update policy. We hope our work will promote such efforts in the DNS standards community.

5.2. Current Defense Implementations

As we have shown in the section 3, although the very popular DNS implementation (BIND) and most of the public DNS servers we tested are vulnerable, three implementations: MaraDNS (version Deadwood-3.0.03), Microsoft DNS (version Windows Sever 2008 R2) and Unbound (version 1.4.11), are immune to the ghost domain attack. The immunity of the latest version of Microsoft DNS derives from a new feature called DNS cache locking [12], but we cannot know the details of this feature because of its proprietary implementation. We reviewed the other two implementations and it turns out that each of them implemented one of the above proposed solutions. Since there is no prior public disclosure of the ghost domain behavior, we do not know whether these two versions of DNS implementation intentionally address the ghost domain name problems or not. Nevertheless, we summarize our findings on these implementations as follows: MaraDNS, has already applied the first solution listed in the above section. It only accepts a zone's delegation data from its parent zone. The Unbound DNS server adopts the 3rd solution that allows overwriting of delegated data but keeps its old TTL value in the cache.

5.3. Delegation Semantics in DNSSEC

We also consider the implication of the ghost domain attack on the DNSSEC system, and we believe a fully deployed DNSSEC is immune to the ghost domain problem. The immunity does not come from the initial intention of DNSSEC using cryptographic signatures to protect the integrity of DNS data. Instead, the immunity is an outcome of a strictly defined delegation behavior. In short, DNSSEC defines a new RR type, `DS` (Delegation Signer), to form a chain of trust between parent and child zone. In the DNSSEC standard, the specification [3] explicitly states that `DS` record can only be obtained from the parent zone. Therefore a ghost domain attacker cannot renew `DS` record of the ghost domain by himself. Without a valid `DS` record, the trust chain will be broken, so haunted security-aware resolvers will only resolve the ghost domain as non-authentic data.

However, in an environment of partial DNSSEC deployment, a security-aware resolver could still be haunted to resolve a ghost domain as authentic results. The reason is that partial deployment DNSSEC raises a trust anchor management issue called "isolated DNSSEC islands" [26]. Without fully DNSSEC deployment, A security-aware resolver cannot validate DNS data from isolated DNSSEC-enabled zones with one single trust anchor from the root. Instead, it needs to be configured with third-party trust anchor providers, such as DNSSEC Look-aside Validation (DLV) providers [33] [34] or public trust anchor lists [24], in order to obtain `DNSKEY` records of isolated DNSSEC-enabled zones to be able to authenticate their DNS data. The attacker can register `DNSKEY` records of the ghost domain to DLV providers and public trust anchor lists, so that DNS data of the ghost domain could still be validated by those trust anchors, even though the `DS` record is deleted from the parent zone. To prevent this, DLV providers and public trust anchor lists need to sync their database with DNS registries in a timely manner. In other words, a malicious domain not only needs to be revoked from the DNS registry, it also should be revoked simultaneously from third-party trust anchor providers. We are currently investigating revocation behaviors of several third-party trust anchor providers.

6. Related Work

Study of Malicious Domain Names. Our work is led by an initial motivation of understanding the lifetime of malicious domains and the effects of domain takedown. In previous studies [22] [23], Moore et. al. showed that most of phishing domains stay alive for several tens of hours before being taken down. The ghost domain problem could make the effect of takedown unpredictable.

Malicious domains must be identified first before being taken down. Recent research has proposed many approaches to distinguish malicious domain names from benign DNS usage. These approaches include extracting various features of malicious domain names from the usage of RR records [15], leveraging registration information [14], passive access logging [29] [1] [6] and lexical construction [19] [35]. From an intrusion detection perspective, the ghost domain exploitation is detectable as it has unusual usage of DNS records.

Cache Poisoning Attacks and Countermeasures. The ghost domain vulnerability comes from the loosely defined sanity check of DNS cache. This weakness is also being exploited by DNS cache poisoning attacks. As early as 1990, Steve Bellovin had indicated that a malicious DNS server can pollute cache resolvers by piggybacking arbitrary records in referral sections [5]. In response, the credibility rule and the bailiwick rule were proposed [32], and then adopted by most of DNS implementations. However, these rules are still insecure and recently have been exploited by the Kaminsky-class cache poisoning attack[18]. After disclosure of the Kaminsky-class attack, a number of approaches were proposed to increase DNS forgery resistance [10] [9] [16] [28], but only a few studies were concerned about the weakness of the DNS cache update policy. Son et. al. [31] gave a formal study of the bailiwick rule and the credibility rule; this work helped us to clarify some details of these rules.

DNS Cache Inconsistency. To some extent, the ghost domain problem is a form of DNS cache inconsistency. As DNS only supports a weak cache consistency by using TTL to limit the lifetime of cached copies, authoritative servers cannot propagate data changes to resolvers in a timely way, failing completely in the ghost domain case. Previous DNS studies have proposed a few approaches to address this problem. DNScup [8] proactively pushes data changes from authoritative server to cache resolvers. Osterwail et. al. proposed Zone State Revocation [25], which embedded `DNSKEY` revocation in DNS response to notify resolvers. Such cache consistency mechanisms could potentially avoid the ghost domain problem. However, considering the critical role of DNS, such a change needs to be carefully evaluated.

DNS Misconfiguration. In [17], Kalafut et. al. presented an interesting phenomenon named *orphan DNS server*. An orphan DNS server is a DNS server which has an address record in the DNS, even though its parent domain does not exist. Orphan DNS servers and ghost domains are superficially similar as both of them resolve domain names that should not exist. But they are substantially different. While orphan DNS servers come from typographical errors and misconfiguration in top level domain zone files, ghost domains are more fundamentally derived from the ambiguously defined DNS cache update policy.

7. Conclusion

In this paper, we present a vulnerability in DNS cache update policy, which prevents effective domain name revocation. Attackers could cause a malicious domain name to be continuously resolvable even after the delegated data has been deleted from the domain registry and after the TTL associated with entry supposed expires. These deleted but resolvable domains are called ghost domain names.

Although we have not found evidence that the vulnerability has been used by previous malicious attacks or botnets, our test results show that the majority of public DNS servers and implementations are vulnerable. Our experiments have also demonstrated that a large scale exploitation of this vulnerability is practical. This vulnerability can potentially allow a botnet to continuously use malicious domains which have been identified and removed from the domain registry. The same vulnerability also potentially allows attackers to make a malicious domain appeared to be deleted at most of the DNS servers but still resolvable at specifically targeted DNS resolvers. This makes the detection of ghost domains even more difficult.

We recommend that the DNS community apply a strict bailiwick rule to fix this vulnerability. Several DNS implementations have adopted various defense mechanisms, but many popular implementations are still vulnerable. Our ongoing work includes implementing patches for open source DNS implementations and addressing possible performance and management issues related to the implementation of a strict DNS cache update policy.

Acknowledgments

We would like to especially thank Jason Gustafson for valuable discussions and thoughtfully proofreading this material, as well as Roberto Perdisci and Zhankao Wen for assistance with experimental resource. We also gratefully thank the anonymous reviewers for their valuable comments, and Jianwei Zhuge, Meng Hu, Zongxu Zhao, Pengfei Xu, Xun Lu and Kevin Warrick for their suggestions and feedback. This work was supported in part by the National Basic Research Program of China (973 Project) grant 2009CB320505. Kang Li's research on this work was partially supported by the USA National Science Foundation (NSF) CISE grant 1127195 and a gift from Cisco systems. Jun Li was partially supported by the 2010 Chair of Excellence award from the University Carlos III of Madrid (UC3M), Spain, and the NSF CAREER award under award no. CNS-0644434. 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 funding agencies.

References

[1] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *19th Usenix Security Symposium*, 2010.

[2] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirement. *RFC4033*, 2005.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. *RFC4034*, 2005.

[4] D. Atkins and R. Austein. Threat Analysis of the Domain Name System (DNS). *RFC3833*, 2004.

[5] S. M. Bellovin. Using the Domain Name System for System Break-ins. In *Proceedings of the 5th conference on USENIX UNIX Security Symposium - Volume 5*, pages 18–18, Berkeley, CA, USA, 1995. USENIX Association.

[6] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. *Proceedings of Network and Distributed Security Symposium (NDSS'11)*, 2008.

[7] R. Boscovich. Taking Down Botnets: Microsoft and the Rustock Botnet. `http://blogs.technet.com/b/microsoft_on_the_issues/archive/2011/03/18/taking-down-botnets-microsoft-and-the-rustock-botnet.aspx`, 2011.

[8] X. Chen, H. Wang, S. Ren, and X. Zhang. Maintaining Strong Cache Consistency for the Domain Name System. *IEEE Transactions on Knowledge and Data Engineering*, 19:1057–1071, 2007.

[9] D. Dagon, M. Antonakakis, K. Day, X. Luo, C. Lee, and W. Lee. Recursive DNS Architectures and Vulnerability Implications. In *Proceedings of Network and Distributed System Security Symposium (NDSS'09)*, 2009.

[10] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee. Increased DNS Forgery Resistance Through 0x20-bit Encoding: security via leet queries. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 211–222. ACM, 2008.

[11] D. Dagon, N. Provos, C. Lee, and W. Lee. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Proceedings of Network and Distributed Security Symposium (NDSS'08)*, 2008.

[12] M. DNS. DNS Cache Locking. `http://technet.microsoft.com/en-us/library/ee683892(WS.10).aspx`.

[13] R. Elz and R. Bush. Clarifications to the DNS specification. *RFC2181*, 1997.

[14] M. Felegyhazi, C. Kreibich, and V. Paxson. On the Potential of Proactive Domain Blacklisting. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'10, pages 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[15] T. Holz, C. Gorecki, K. Rieck, and F. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *Proceedings of Network and Distributed Security Symposium (NDSS'08)*, 2008.

[16] J. G. Hy. Anti DNS Spoofing-Extended Query ID (XQID). `http://www.jhsoft.com/dns-xqid.htm`, 2008.

[17] A. J. Kalafut, M. Gupta, C. A. Cole, L. Chen, and N. E. Myers. An Empirical Study of Orphan DNS Servers in the Internet. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 308–314, New York, NY, USA, 2010. ACM.

[18] D. Kaminsky. Its the end of the cache as we know it. *BlackHat USA*, 2008.

[19] J. Ma, L. Saul, S. Savage, and G. Voelker. Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM, 2009.

[20] P. Mockapetris. Domain Names - Concepts and Facilities. *RFC1034*, 1987.

[21] P. Mockapetris. Domain Names - Implementation and Specification. *RFC1035*, 1987.

[22] T. Moore and R. Clayton. Examining the Impact of Website Take-Down on Phishing. In *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, eCrime '07, pages 1–13, New York, NY, USA, 2007. ACM.

[23] T. Moore and R. Clayton. The Consequence of Non-Cooperation in the Fight against Phishing. In *eCrime Researchers Summit, 2008*, pages 1–14. IEEE, 2008.

[24] E. Osterweil, D. Massey, and L. Zhang. Deploying and monitoring dns security (dnssec). *Computer Security Applications Conference, Annual*, 0:429–438, 2009.

[25] E. Osterweil, V. Pappas, D. Massey, and L. Zhang. Zone State Revocation for DNSSEC. In *Proceedings of the 2007 workshop on Large scale attack defense*, LSAD '07, pages 153–160, New York, NY, USA, 2007. ACM.

[26] E. Osterweil, M. Ryan, D. Massey, and L. Zhang. Quantifying the Operational Status of the DNSSEC Deployment. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 231–242, New York, NY, USA, 2008. ACM.

[27] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang. Impact of Configuration Errors on DNS Robustness. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 319–330. ACM, 2004.

[28] R. Perdisci, M. Antonakakis, X. Luo, and W. Lee. WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning

Attacks. In *IEEE/IFIP International Conference on Dependable Systems & Networks, DSN'09.*, pages 3–12. IEEE, 2009.

[29] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In *Annual Computer Security Applications Conference*, volume 0, pages 311–320, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[30] G. Sisson. DNS SURVEY. `http://dns.measurement-factory.com/surveys/201010/`, 2010.

[31] S. Son and V. Shmatikov. The Hitchhikers Guide to DNS Cache Poisoning. In *Security and Privacy in Communication Networks*, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 466–483. Springer Berlin Heidelberg, 2010.

[32] P. Vixie. DNS and BIND Security Issues. In *Proceedings of the 5th conference on USENIX UNIX Security Symposium - Volume 5*, pages 19–19, Berkeley, CA, USA, 1995. USENIX Association.

[33] P. Vixie. Preventing Child Neglect in DNSSECbis Using Lookaside Validation(DLV). *IEICE Transactions on Communications*, pages 1326–1330, 2005.

[34] S. Weiler. DNSSEC Lookaside Validation (DLV). *RFC5074*, 2007.

[35] S. Yadav, A. Reddy, A. Reddy, and S. Ranjan. Detecting Algorithmically Generated Malicious Domain Names. In *Proceedings of the 10th annual conference on Internet measurement*, pages 48–61. ACM, 2010.