Passive DNS Replication

Florian Weimer <fw@deneb.enyo.de>

April 2005

Overview

The domain name system (abbreviated 'DNS') provides a distributed database that maps domain names to record sets (for example, IP addresses). DNS is one of the core protocol suites of the Internet. Yet DNS data is often volatile, and there are many unwanted records present in the domain name system. This paper presents a technology, called *passive DNS replication*, to obtain domain name system data from production networks, and store it in a database for later reference.

The present paper is structured as follows:

- Section 1 briefly recalls a few DNS-related terms used throughout this paper.
- Section 2 motivates the need for passive DNS replication: DNS itself does not allow certain queries whose results are interesting in various contexts (mostly related to response to security incidents).
- Section 3 describes the architecture and of the `dnslogger` software, an implementation of passive DNS replication.
- In section 4, successful applications of the technology are documented.

1 DNS terminology

This section provides a very brief sketch of DNS. The terminology presented here will be used in later sections. Readers who are not familiar with the terms are encouraged to ask their local DNS operator, or consult a reference manual such as [AL01].

DNS data is divided into *zones*. Each zone is served by a set of *authoritative name servers*. Authoritative name servers provide *authoritative answers* for data contained in the zones they serve. (The concept of authority implies that these servers do not contact other name servers to include data in replies which is not available locally.)

A second type of name server is the *resolver*. Resolvers can only return *non-authoritative answers* to clients. They start at the root servers and follow zone delegations (processing the authoritative answers), until they reach the final authoritative name server for the correct zone. Aggressive caching makes this process run fast, however stale data (which is no longer available from any authoritative name server) can be returned to clients.

DNS only supports a single kind of query: given a domain name and a record type, all matching records are returned. Other search keys must be converted to a domain name before they can be used in a DNS query. The most common example are reverse lookups for IP addresses.

2 The need for additional DNS query types

The initial motivation for the development of passive DNS replication was the inadequacy of PTR-based reverse lookup, which maps IP addresses to domain names. In general, the data source for PTR answers is just another zone which is not automatically updated when someone adds a new host name for an IP address covered by the reverse zone. (Of course, DNS cannot guarantee this due to its distributed nature: the A record can be located in any zone, served by authoritative servers which are different from the servers that provide the reverse zone.)

As mentioned at the end of the previous section, DNS only supports a single query. Anyone can add DNS records to a zone he or she controls, and new zones can be created easily: Many registrars for second-level domains offer freely editable zone files. Yet there are no safeguards which ensure that the resource records only point to infrastructure (IP address space, domain names) which belongs to the zone owner.

However, once the data has been stored in a local database, more elaborate queries are possible, which leads to further applications.

2.1 Malware containment

Malware often contains a hard-coded domain name which identifies a command and control host. The malware performs a lookup on this domain name to obtain a set of IP addresses, and contacts one of those servers. After that, it waits for incoming commands, and performs the requested actions (for example, scanning for more vulnerable hosts, or flooding a specified target with garbage packets).

Even if the malware is still operational on the victim's computer, some of its functionality is unavailable once the domain name has been removed from DNS. Therefore, knowledge of the domain name is important, otherwise it is impossible to contact a DNS administrator with a request for removal. In addition, if the domain name is known, all of its associated IP addresses can be filtered locally, which helps to contain the malware infection within the local network.

The problem is that malware is typically detected *after* it has performed its domain name lookup. Even if it is possible to eavesdrop on the network traffic (which is technically infeasible in most service provider environments), the network traffic does not reveal the domain name. Only during reconnection after disruption or similar events, recovery of the domain name is possible. This adds a significant delay, which is sometimes unacceptable.

2.2 Trademark protection

In most jurisdictions, trademarks must be defended against (deliberate or accidental) infringement, otherwise they dilute and finally lose their status as trademark. DNS zone data can be examined for potential infringement.

In order to cut down the rate of false positives (e. g. domain names which are held by the trademark owner, but not used officially), the name of the name servers of those domains (as given in NS resource records) can be used. If the servers belong to the trademark owner, the

company very likely also owns the domain. IP addresses can also be taken into consideration and compared to the address ranges normally used by the company.

This approach does not use any out-of-band data and is not affected by the poor data quality often found in those resources. For example, for some top-level zones, domain name WHOIS information is in a notoriously bad state and lots of entries are unmaintained or contain obviously forged data. Zone data, which is actually used for production purposes, is generally more correct and up-to-date, although it might lack details that (in some cases) are available in WHOIS registries.

2.3 Phishing

Much in the same way, some forms of "phishing attacks" can be detected. In these attacks, someone creates a web site which looks like the official site of the attacked company, under an official-looking domain name. The web site, completely operated by the attacker, collects personal information, such as account names and passwords. Later, the attacker uses the collected data to defraud the attacked company and its customers. Of course, the attacker does not have to use domain names which resemble official ones used by the company, and detection of the attack does not stop it. However, passive DNS replication can be used as a building block in a broader defense against such attacks.

2.4 Analysis of IP-based filters

If other methods have uncovered evidence that a particular IP address at another network behaves in particularly bad way (if it hosts a phishing site, for example), a glance at archived DNS data can show that the IP address in question is used by multiple different services. A network operator can assess the collateral damage before applying an IP-based filter.

Similarly, anti-censorship activists can use this information to support their argument that IP-based filters are often too broad and unwarranted.

2.5 MX theft and other policy violations

MX theft occurs when someone points an MX record to a loosely-configured external mail server, without proper authorization, and uses it as a backup mail relay for this domain. (This differs from a completely open mail relay. Most mail server software offers a configuration option that allows it to serve as a mail relay for all domains that have an MX record that point to the local host. In the past, this has been used to significantly simplify large mail setups.)

While MX theft is not a real issue on the current Internet, other forms of policy violations are possible, especially on relatively open university or corporate networks. For example, additional web servers are installed, and domain names are pointed to them, without authorization from the responsible staff.

Passive DNS replication can recover most of the actively used DNS records pointing to one's own network resources, and thus support enforcing particular policies.

2.6 Recovery of zone data

If a some catastrophic failure causes an important zone to vanish from the public DNS, it is often desirable for other service provides to resurrect it, if only to keep down the number of complaints from their own customers.

Today, the most likely failures of this kind are a corrupted master copy which is propagated to all authoritative name servers of the zone, and an unauthorized delegation change.

If resource records are stored together with time-stamp information, it is possible to recover the view of a zone at a certain date. In particular, incorrect records (maliciously or inadvertently) added later can be discarded.

2.7 Replacement of documentation

Some network operators are required by law to provide a list of domain names used by them. In Germany, this mostly applies to networks in the public sector and is the result of an obscure combination of legal requirements and blanket authorization of certain government bodies.

Open university networks may have documentation for official domain names which have been centrally registered, but if anyone can run his or her own authoritative name server (both technically and in accordance with university policy), centralized documentation is very likely incomplete.

Passive DNS replication can provide a better approximation of the situation. It remains to be seen if the authorities accept them, though.

2.8 Less desirable applications

We should not forget that DNS data in bulk form can be abused. The spamming industry shows a lot of interest in domain name lists. Such lists can be used as a starting point for all kinds of crawlers. Public access to live DNS data raises some privacy concerns as well.

3 An implementation of passive DNS replication

In the previous section, we presented applications for which locally stored DNS records are desirable. In this section, we describe an implementation of passive DNS replication, called `dnslogger`, which obtains DNS data from a production network and archives it.

3.1 Data sources

Before it is possible to query a local database of DNS data, the data has to be record by some means. There are several possibilities to obtain DNS data in larger quantities:

- You can periodically *poll DNS records* and gather this way (`dnswatch` [Kri04] is a tool which supports this). The disadvantages are obvious: You must already know what you are looking for. There are some scalability issues, too, and systematically querying DNS records raises some suspicion from DNS operators.

- Arrange for *zone file transfers* (not necessarily using the standard DNS mechanism). This requires cooperation from those who maintain the master copy. Some DNS operators offer zone file access (ICANN requires that gTLD operators publish zone files, for example), but others do not. TLD zones like the `.com` zone contain only the nanmes of second-level domains, and names and IP addresses of name servers, but not regular host addresses, so that it is still necessary to guess domain names and actively gather further information by querying the listed name servers.

- You can reconfigure your resolvers to *log queries*. The drawback is that you have to reconfigure each server individually, and that it does exactly that: it logs queries, which means that the data is related to customer IP addresses, so there are real privacy concerns to address. Furthermore, if you are more interested in DNS data than in customer behavior, there is a considerable redundancy in query logs. (One advantage of customer-related logs is that you can alert customers whose machines generate suspicious DNS requests.) An important aspect of query logs is that you do not need to know in advance what are you trying to find.

- *Capture DNS packets* on the network. This is what `dnslogger` does. It combines the advantages of query logging with a powerful approach to data reduction, which allows to collect DNS data on networks with tens of thousands of hosts, using only a modest investment in hardware.

We called the latter approach *passive DNS replication* mainly to avoid more controversial terms such as *sniffing*, *monitoring* or even *DNS data retention*. The *passive* part is intended to reflect the fact that no cooperation from zone owners is necessary, and that no additional DNS requests are generated.

Compared to the approach based on zone files, there is an important difference: we can never be sure that our data is complete. However, if passive DNS replication is used to support mostly local decisions, this is not a significant problem in most cases: there is no customer interest anyway in records which are missing. Furthermore, a lot of domains are simply dormant and unused, and they are included in zone files.

3.2 The architecture

Figure 1 shows the `dnslogger` architecture. Each rectangle is a separate process (processes are distributed across multiple hosts, as described below). The purpose of these processes is as follows:

- A *sensor* captures DNS packets on the network. It applies some filters (for example, it might only forward authoritative answers), and forwards the remaining packets to an analyzer. Two independent implementations exist, one in Perl and one in rigorously tested C (called `dnslogger-forward`).

- The *analyzer* parses the DNS packets and extracts the data which should be processed further (domain names, IP addresses, and so on).

- The *collector* takes the analyzer output and updates the database which is used for DNS data archival.
- The *requery daemon* issues queries for certain DNS records identified by the analyzer (see section 3.5).
- A *query processor* waits for user-supplied queries and executes them on the database. `dnslogger` provides a WHOIS server and a command-line tool which runs on the collector host.

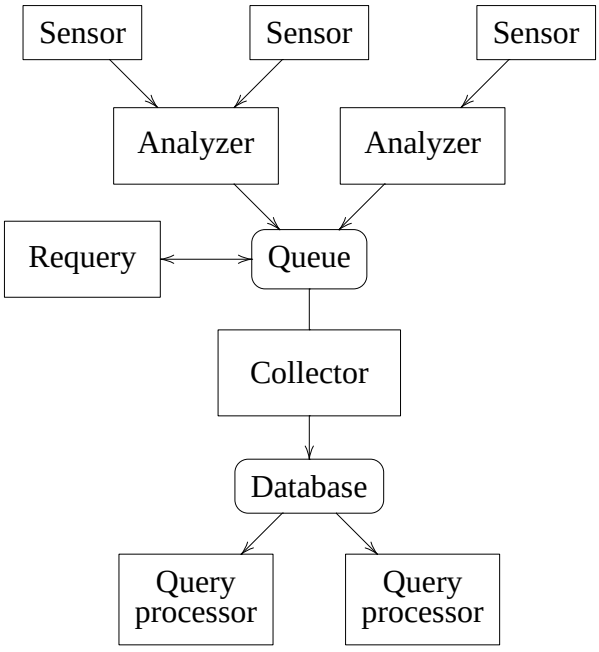

Figure 1: `dnslogger` architecture

Sensors are distributed across the network. (Analyzers can be distributed across multiple hosts as well.) The collector and query processor run on the same host, but multiple collectors per host are possible, and the WHOIS query processor can combine their databases.

Both the sensor and the analyzer face soft real-time requirements. Once the data has passed the analyzer, a sufficient average processing is sufficient. For example, the `dnslogger` collector processes updates in batches, to reduce the number of flushes to stable storage which are required.

Berkeley DB [Sle04] is used as the underlying database technology. The sensors are implemented in Perl and C, as mentioned above. The remaining parts are implemented in Ada.

Some applications could benefit from ad-hoc queries, which would be supported in a much better way by an SQL database. However, experiments showed that PostgreSQL does not offer the required performance on the available hardware. If faster hardware is available (or if the data set is smaller), switching to an SQL database looks like a good option.

3.3 Verification

What is a valid DNS packet? This question is surprisingly hard to answer.

Most DNS communication is transmitted using UDP. The only protection against blindly spoofed answers is a 16 bit message ID embedded in the DNS packet header, and the number of the client port (which is often 53 in inter-server traffic). What is worse, the answer itself contains insufficient data to determine if the sender is actually authorized to provide data for the zone in question. In order to solve this problem, resolvers have to carefully validate all DNS data they receive, otherwise forged data can enter their caches [CC96, CC97, Hav00, Hav01].

DNSSEC [AAL+05] still awaits deployment, and DNS itself totally lacks any cryptographic protection. An attacker who can send both requests and answers to a sensor is likely to fool even the most advanced DNS data validation techniques.

Consequently, `dnslogger` sidesteps these issues and simply does not verify any data. This has to be taken into account when using its output, of course. On the other hand, if a sensor captures an obviously bogus DNS response, we cannot exclude the possibility that this response has entered a resolver cache and is processed by a host. Therefore, it makes some sense to store bogus responses.

If bogus data is injected in the `dnslogger` database in large quantities, a rollback is possible based on the recorded time stamp information (see the next section).

3.4 The data reduction process

As mentioned briefly in section 3.1, the collector involves a data reduction step. This step is necessary because the raw stream of DNS response packets which arrives from the sensors contains a lot of redundant information and is not suitable for direct storage.

- Only DNS resource records are stored. The information which records were in the same DNS response packet is lost.
- For each DNS record, two time stamps are recorded in the database: one describes the time of the first occurrence of the record, the other the time of the last occurrence.
- Time-to-live (TTL) fields and DNS record classes are not stored. TTL values have only a meaning if they are contained in an authoritative answer, in specific sections. The class field is practically unused because all records actually used by applications have class IN, and most IANA-assigned DNS record types are not class-specific.
- An experimental feature records the IP address of the authoritative name server which sent a particular record. A reliable implementation is quite difficult, however. On the one hand, we want to store source addresses for all interesting records, on the other, we want to avoid labeling caching resolvers as authoritative for the records they return. Unfortunately, mixed authoritative and caching servers do not unambiguously indicate which parts of a response are authoritative. Instead, `dnslogger` uses an experimental heuristic which marks most data which does not exactly match the question section of a DNS response packet as non-authoritative.

A consequence of the time stamp approach is that all database updates caused by new DNS responses are *idempotent*: it does not matter if they are performed just once, or multiple times. Furthermore, they are *commutative*: the order in which they are applied does not matter. This means that updates can be batched and reordered as necessary. The `dnslogger` software uses these freedoms to increase its performance, and can process thousands of DNS records per second.

However, after several weeks of continuous operation, the working set in the database still tends to exceed the available RAM typically available on the PC platform (between 1 GB and 2 GB). `dnslogger` installations can address this problem by switching databases periodically. The WHOIS database front end is able to merge record sets from all databases, and return it as if there was just a single database. This approach is reminiscent of [MOPW00], but idempotent and commutative updates make the implementation much simpler (and arguably less efficient).

In a previous version of the `dnslogger` software, another attempt was made at data reduction: all domain names were converted to a 32 bit domain ID. However, this reduced access locality, and multiple disk seeks were required for each record returned in a query. Even though query performance was not a primary goal, query performance degraded so much that it became unacceptable. The current database scheme is highly denormalized and structured around a single B+-tree which holds domain names (in reversed form, to increase locality). IPv4 and IPv6 lookups are processed using separate indices, and yet another index is built on top of domain names in their unreversed form. Insert performance is comparable to the old version (or even better, when the working set size approaches the size of available memory). In the end, even though each domain name is stored multiple times (instead of shorter IDs), disk space requirements did not increase significantly, probably due to Berkeley DB's key prefix compression.

3.5 Truncated responses

If a query to an authoritative name server matches a large number of records (and the response UDP packet would exceed 512 bytes, the limit set by the DNS standards), some servers, instead of supplying only partial information, return an empty response with a set truncation (`TC`) bit. In response, the caching name server which queries for the resource record will issue a second query, this time over TCP. Some forms of DNS query validation used to combat denial-of-service attacks also work with truncated responses to trigger additional queries [PTGA03].

Unfortunately, such truncated, empty responses cause problems for completely passive DNS replication, unless capturing TCP queries and responses (including some limited form of TCP stream reassembly) is implemented. The latter is very complex, therefore `dnslogger` uses a different approach: The analyzer extracts the question section of truncated DNS responses, and passes them to the requery daemon. The requery daemon queries a standard DNS resolver over a TCP connection, decodes the response, and passes it to the collector.

3.6 Sensor placement

Sensors should be placed close to large, caching name servers, or at uplinks of networks containing caching name servers. Typically, a caching name server requests much more diverse data from other name servers than a single authoritative name server can provide. It makes sense to deploy sensors close to authoritative name servers only if above-than-average coverage of the zones served by those servers is desired.

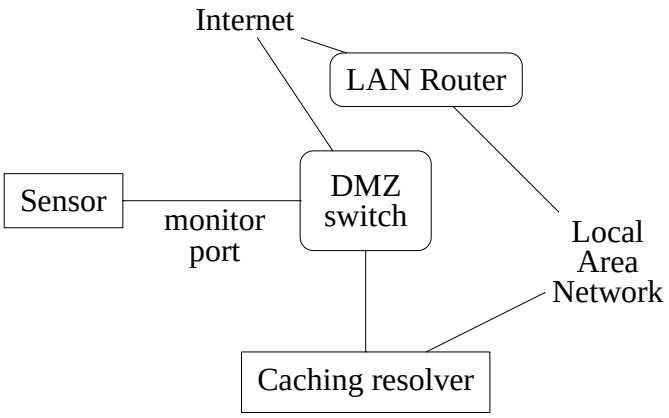

Figure 2: Sensor placement

Furthermore, as shown in Fig. 2, the sensor is best placed at a point where it does not catch traffic between the caching resolver and its clients. Here, the sensor captures packets sent to and from a caching name server on a monitor port of the DMZ switch. In such setups, we are not interested in DNS responses sent to the LAN because the resource records have already been recorded when the entry was cached for the first time.

A similar effect can be achieved using IP-based filters in the sensor, to restrict it to traffic that flows in a particular direction. `dnslogger-forward` also provides the option to discard non-authoritative answers.

4 Examples and results

4.1 Botnet mitigation

Several CSIRTs are using collected DNS data to identify botnet controllers and obtain secondary addresses of these hosts.

This works reasonably well on a local scale, but global botnet mitigation is hampered by the synchronization problem: both the DNS records and the current controller hosts must be removed at approximately the same time, otherwise the botnet owner can just alter the DNS records to a new host, or use the current controller to instruct the bots to download new malware with a different domain name.

4.2 The `kimble.org` fiasco

The original Blaster worm performed a denial-of-service attack on the IP address of the domain `windowsupdate.com`, which is owned by Microsoft. In order evade the attack, Microsoft removed the A record from DNS.

Blaster.E is a variant of the Blaster worm which attacks `kimble.org`. Table 1 shows that someone regularly changed the usual IP address of 127.0.0.1 (which is a harmless setting) to a globally routed IP address. (As you can see, the collector database started its operation on June 23. In the tables of this section, we omit the 'last seen' time stamp.)

First seen	Domain name	Type	Data
2004-06-23 13:58:51	kimble.org	A	127.0.0.1
2004-08-07 16:14:00	kimble.org	A	207.234.155.17
2004-10-20 07:15:58	kimble.org	A	212.100.234.54
2004-10-20 16:12:56	kimble.org	A	64.203.97.121
2004-10-21 17:15:01	kimble.org	A	212.113.74.58
2004-10-21 17:45:01	kimble.org	A	195.130.152.100
2004-10-31 14:45:01	kimble.org	A	195.225.218.59
2004-11-02 23:15:01	kimble.org	A	206.132.83.2
2004-11-04 18:15:01	kimble.org	A	213.139.139.206
2004-11-21 03:15:02	kimble.org	A	216.7.173.212
2004-11-25 22:45:02	kimble.org	A	38.112.165.60

Table 1: IP address of `kimble.org`

In this case, passive DNS replication led to a rediscovery of the Blaster.E problem, based on reports of the more recent attacks. Subsequent victims were informed about the nature of those attacks.

Unfortunately, the Internet community has yet to develop policies to deal with such 'tainted' domain names. Obviously, even the legitimate owner should not be allowed to change DNS records in such harmful ways.

4.3 Collateral damage of IP-based filters

Passive DNS replication has been used to assess the collateral damage of IP-based filters. Such filters are usually the only scalable tool ISPs have at hand to remove unwanted web content (such as hate speech, pornography, or phishing sites). In some cases, such filters are recommended by government bodies.

For example, in November 2004, T-Com apparently injected a null route into its IGP to block access to a phishing site which targeted T-Com customers. Unfortunately, the phishing site was located on a web server which also hosted completely harmless content on the same IP address (but under different domain names, on other virtual servers).

Currently, collected DNS data is mainly used after the fact, to stress that IP-based filters cause too much harm themselves and cannot be the final answer to harmful content on the web. It is desirable, though, that DNS data is used at the filter planning stage, to minimize collateral damage.

4.4 ebay.de domain transfer

On 2004-08-28, Ebay's domain service provider for the DE zone implicitly expressed consent to a request that DENIC, the DE registry, should transfer the delegation for ebay.de to a new ISP. This is clearly visible in the recorded DNS data in table 2.

First seen	Domain name	Type	Data
2004-06-23 08:21:57	ebay.de	NS	crocodile.ebay.com
2004-06-23 08:21:57	ebay.de	NS	sjc-dns1.ebaydns.com
2004-06-23 08:21:57	ebay.de	NS	sjc-dns2.ebaydns.com
2004-08-28 05:34:01	ebay.de	NS	ns1.goracer.de
2004-08-28 05:34:01	ebay.de	NS	ns2.goracer.de

Table 2: Changed ebay.de delegation

The older, correct data is still available in the database, and an interested ISP could use it to restore the correct DNS view for its own customers. (In this case, it was sufficient to restart the resolvers to remove stale data from the cache because DENIC performed an emergency zone update to restore the old delegation.)

4.5 DNS data is extremely localized

The author was surprised to discover how sensor-specific the collected DNS data is. Apparently, Internet usage has become much less U.S.-centric when it reached mainstream. A particular site only observes rather localized part of the whole DNS system.

Content distribution networks are another interesting development. Most of them run special authoritative name servers which return different IP addresses, depending on the location of the resolver sending requests to them. As a result, DNS data is no longer globally consistent, and sensors can only capture a specific view.

This means that service providers who want to use passive DNS replication to support IP blacklist planning must set up their own sensor (and analyzer and collector, if they cannot contribute to an existing collector for legal reasons). The DNS data which has been collected on other networks likely does not reflect the needs of their customer base.

4.6 Privacy Implications

Two different kinds of personally identifiable information arises in the context of domain names: data about the circumstances of the query (IP addresses involved, time of query, the domain name requested), and the actual contents of a domain name.

The first issues are easily addressed by the placement of the sensors. If the guidelines outlined in section 3.6 are followed, the sensor only observes inter-server traffic. This means that end user IP addresses cannot be recovered at that point. Thanks to caching, most queries do not result in inter-server queries, which further reduces the potential for misuse. Using flow data [NNW00], it might be possible to correlate the time-stamp information and server

IP address information in both data sets and thus obtain the end user addresses for some non-cached queries, but this privacy invasion is more a result of flow logging than of our DNS replication effort.

The second set of issues is harder to dismiss. There are a few web service providers which use wildcard A records and session IDs embedded into domain names to implement user tracking. In one case, these domain names are embedded in "web bugs" and are used to track users without their consent, so the publication of these domain names does not do any harm (because it dilutes the user identification, which is increasing privacy in this case). In other cases, the domain name contains a session ID within a web application.

As a consequence, the publicly accessible database front end mentioned in the next sections delays the availability of new resource records by 20 minutes. It is expected that after this time period, potentially affected sessions have expired.

4.7 RUS-CERT's front end

RUS-CERT offers a publicly accessible front end to their collector database at:

`http://cert.uni-stuttgart.de/stats/dns-replication.php`

To prevent abuse, this front end only supports a subset of the queries supported by the underlying database, and the number of records which are returned in response to a query is limited. The NSP security community has access to the more powerful WHOIS front end [Wei05].

Conclusion

In this report, we motivated the need for passive DNS replication, described an architecture and an implementation, and presented observations. Hopefully, the technology is used more widely in the future, especially in the planning stage of IP-based filters.

Future work areas include refinements in source address recording, and a distributed WHOIS server which can directly access databases on multiple hosts. Fragmented EDNS0 responses [Vix99] raise issues similar to TCP responses which still need to be addressed.

Acknowledgments

The author wishes to thank the NSP security community whose feedback helped to improve the `dnslogger` service.

RUS-CERT kindly provides resources to run the implementation of passive DNS replication described in this report. Several third parties donate DNS response data.

References

[AAL+05] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. DNS Security introduction and requirements. RFC 4033, Internet Society, March 2005. `http://www.rfc-editor.org/rfc/rfc4033.txt`.

[AL01] Paul Albitz and Cricket Liu. *DNS and BIND*. O'Reilly, April 2001.

[CC96] CERT Coordination Center. CERT Advisory CA-96-02, February 1996. `http://www.cert.org/advisories/CA-96.02.bind` (2004-12-29).

[CC97] CERT Coordination Center. CERT Advisory CA-1997-22, August 1997. `http://www.cert.org/advisories/CA-1997-22.html` (2004-12-29).

[Hav00] Jeffrey S. Havrilla. Microsoft Windows domain name resolver service accepts responses from non-queried DNS servers by default, April 2000. `http://www.kb.cert.org/vuls/id/458659` (2004-12-29).

[Hav01] Jeffrey S. Havrilla. Microsoft Windows NT and 2000 domain name servers allow non-authoritative RRs to be cached by default, June 2001. `http://www.kb.cert.org/vuls/id/109475` (2004-12-29).

[Kri04] John Kristoff. dnswatch, September 2004. `http://aharp.ittns.northwestern.edu/software/` (2004-12-28).

[MOPW00] Peter Muth, Patrick O'Neil, Achim Pick, and Gerhard Weikum. The LHAM log-structured history data access method. *The VLDB Journal*, 8:199–221, 2000.

[NNW00] John-Paul Navarro, Bill Nickless, and Linda Winkler. Combining Cisco NetFlow exports with relational database technology for usage statistics, intrusion detection, and network forensics. December 2000. `http://www.usenix.org/publications/library/proceedings/lisa2000/full_papers/navarro/navarro_html/`.

[PTGA03] Guy Pazi, Dan Touitou, Alon Golan, and Yehuda Afek. Protecting against spoofed DNS messages. United States Patent Application 20030070 A1, United States Patent Office, April 2003.

[Sle04] Sleepycat Software, Inc. Berkeley DB, December 2004. `http://www.sleepycat.com/docs/`.

[Vix99] Paul Vixie. Extension mechanisms for DNS (EDNS0). RFC 2671, Internet Society, August 1999. `http://www.rfc-editor.org/rfc/rfc2671.txt`.

[Wei05] Florian Weimer. Passive DNS replication WHOIS server, March 2005. `http://www.enyo.de/fw/software/dnslogger/whois.html`.