Malicious Payload Distribution Channels in Domain Name System

A Thesis in the Concordia Institute for Information Systems Engineering

Presented in Partial Fulfilment of the Requirements
for the Degree of Master of Applied Science in Information Systems Security
at the Concordia Institute for Information Systems Engineering
Concordia University, Montréal, Québec, Canada

Abdullah Mert Kara

December 2013

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Abdullah Mert Kara**
Entitled: **Malicious Payload Distribution Channels in Domain Name System**

and submitted in partial fulfilment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Andrea Schiffauerova _____ Chair

Dr. Lingyu Wang _____ Examiner

Dr. Peter Grogono _____ Examiner

Dr. Mourad Debbabi _____ Supervisor

Dr. Mohammad Mannan _____ Supervisor

Approved _____
Chair of Department or Graduate Program Director

_____ 2013 _____
Christopher Trueman, Ph.D., Dean
Faculty of Engineering and Computer Science

Abstract

Malicious Payload Distribution Channels in Domain Name System

Abdullah Mert Kara

Botmasters are known to use different protocols to hide their activities under the radar. Throughout the past years, several protocols have been abused and recently Domain Name System (DNS) also became a target of such malicious activities. In this dissertation, we analyze the use of DNS as a malicious payload distribution channel. To the best of our knowledge, this is the first comprehensive analysis of these payload distribution channels via DNS. We present a system to characterize such channels in the passive DNS (pDNS) traffic by modeling DNS query and response patterns. Then, we analyze the Resource Record (RR) activities of these channels to build their DNS zone profiles. Finally, we detect and assign levels of intensity for payload distribution channels by using a fuzzy logic theory. Our work is based on an extensive analysis of malware datasets for one year, and a near real-time feed of pDNS traffic. The experimental results reveal few long-running hidden domains used by Morto worm to distribute malicious payloads. We also found that some of these payloads are in cleartext, without any encoding or encryption. Our experiments on pDNS traffic indicate that our system can detect these channels regardless of the payload format.

Passive DNS is a useful data source for DNS based research, and it requires to be stored in a database for historical data analysis, such as the work we present

in this dissertation. Once this database is established, it can be used for any sort of threat analysis that requires DNS oriented intelligence. Our aim is to create a scalable pDNS database, that contains potentially valuable security intelligence data. We present our pDNS database by discussing the database design, implementation challenges, and the evaluation of the system.

Acknowledgements

As in every successful work, the collaboration and supervision played an important role for me to reach to this point. Throughout my studies, my supervisors Dr. Mourad Debbabi and Dr. Mohammad Mannan taught me the important principles of turning ideas into applications. I would like to take this opportunity to thank them. Also, I would like to send my sincere thanks to Paul Vixie for his valuable comments.

In Computer Security lab, I have had the chance to work with great people, and I thank all of them for their friendships. Also, my labmates who were great companions during this journey; Hamad Binsalleeh, Gaby Dagher, Sahba Sadri. Also, I thank National Cyber-Forensics & Training Alliance (NCFTA) Canada for providing facilities for conducting research, this work would not be possible without their active supports.

The result of studying abroad, my family in Turkey paid the biggest price by me being physically away from them. This research work would be a minimal payment for every moment, that I could not be with them. With all my sincere feelings, I would like to extend my appreciations to my mother Sıttıka, my father Mehmet, my sister Mehtap, and my brother Mustafa. Also, I truly apologize to my niece İpek for leaving her without me during these years. Finally, I thank Claris for giving me support for everything during my studies.

Dedicated to my family and Claris ...

Contents

List of Figures

List of Tables

List of Equations

List of Acronyms

2LD	Second-Level Domain
3LD	Third-Level Domain
API	Application Programming Interface
C&C	Command and Control
ccTLD	Country Code Top-Level Domain
DDoS	Distributed Denial of Service
DGA	Domain Generation Algorithm
DKIM	Domainkeys Identified Mail
DMARC	Domain-based Message Authentication, Reporting and Conformance
DNS	Domain Name System
EDNS	Extension Mechanisms for DNS
FBI	Federal Bureau of Investigation
FIFO	First In First Out
FQDN	Fully Qualified Domain Name
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IRC	Internet Relay Chat

ISP	Internet Service Provider
JSON	JavaScript Object Notation
MAC	Media Access Control
NCFTA	National Cyber-Forensics & Training Alliance
NDA	Non-Disclosure Agreement
OE	Opportunistic Encryption
P2P	Peer-to-Peer
pDNS	passive DNS
RDBMS	Relational Database Management System
RDP	Remote Desktop Protocol
RFC	Request for Comments
RR	Resource Record
SIE	Security Information Exchange
SOA	Start of Authority
SPF	Sender Policy Framework
SQL	Structured Query Language
TLD	Top-Level Domain
TTL	Time to live
UDP	User Datagram Protocol
URL	Uniform Resource locator
VLAN	Virtual Local Area Network

Chapter 1

Introduction

1.1 Motivation

DNS is a part of our daily Internet activities, and it is tightly coupled with any network infrastructure. It is a simple yet powerful database, which holds Internet Protocol (IP) addresses of every domain name. Therefore, Internet users do not need to remember the IP addresses of websites, they only remember a domain name, which is dedicated for a particular website, e.g., `google.com`. DNS is a fundamental part of any activity in the Internet, and it is attracting botmasters to use its facilities to maintain their malicious networks.

DNS is often abused by attackers, and the recent incidents have showed that it is still a vulnerable protocol [72, 73, 43]. Spamhaus[1], a non-profit anti-spam organization, was a target of DNS based Distributed Denial of Service (DDoS) attack. It was the biggest DDoS attack in the history, and it crippled the infrastructure of Spamhaus [72]. Moreover, the attack almost affected the global DNS traffic [72], which straightforwardly affects the Internet all around the globe. In another incident, a malware family, namely DNSChanger, affected more than four million computers in the US alone. The situation became serious that Federal Bureau of Investigation (FBI) took over the case to fight against this profit oriented criminal activity [88]. The

[1] http://www.spamhaus.org/

attack is accomplished by changing the DNS host settings on an infected machine, therefore all DNS queries are redirected to a rogue DNS resolver, which is operated by the botmasters. The whole eradication process required to block infected machines from connecting to the Internet. These attacks show how fragile DNS is, and any attack on DNS can render the Internet almost unusable.

Attackers do not only attack on DNS, they also make it a fundamental part of their malicious networks. Recently, DNS is used by botmasters to breach secure networks to steal sensitive information without alerting network security systems [22]. It is used as a stealth communication channel between bots and Command and Control (C&C) servers. The DNS traffic is often considered to be harmless, and network administrators allow it to bypass any security monitoring. This gives an opportunity to botmasters to send and receive data, even in highly protected networks. Compared to other protocols (e.g., Hypertext Transfer Protocol (HTTP), Internet Relay Chat (IRC), Peer-to-Peer (P2P)) [13, 32, 50], DNS becomes a perfect candidate for covert channels in such networks. Moreover, its naive architecture provides facilities for data transfer. Any existing defense technology against botnet communication channels is rendered useless with this new type of attack. The attack is based on a vulnerability of the DNS protocol, which allows to embed arbitrary strings in DNS query and response packets. Botmasters send the attack payloads in DNS queries, and bots use the same method to talk back to botmasters. In this way, a malicious payload distribution channel is established in DNS. Because the entire communication is established in DNS queries, the communication channel is only traceable in DNS logs, which is often not monitored in a network. Beside, any semantic based analysis on DNS logs might result in a high amount of false positives or negatives. It is required to have an

approach to monitor DNS in a global scale to detect such communication channels, and alert the authorities about domains involved.

Detecting any DNS based threat requires a hands-on experience with real DNS data logs. This type of studies are often done by using local network DNS logs, however local logs cannot be used to look at global trends in malicious activities. It is important to analyze global DNS activities to understand and design defense mechanisms against emerging threats, such as malicious payload distribution channels in DNS. PDNS replication provides a good dataset for such studies. PDNS is a system that is deployed on name servers to replicate DNS queries for different purposes, i.e., security research. For example, pDNS is used to detect botnets by analyzing DNS query anomalies [12]. However, it is a continuous data stream, and we need to have a historical database of pDNS to correlate previous attacks with current ones to model the behaviors of malicious communication channels. An efficient and scalable database solution is required to store this tremendous amount of DNS data.

1.2 Problem Statement

In this dissertation, the detection of malicious payload distribution channels in DNS, and the design and implementation of an efficient and scalable pDNS database are addressed. Additionally, the necessity of having such historical data for DNS based research are discussed.

The main research questions investigated in this dissertation are:

1. ***Question 1:*** Is it possible to model the behavior of DNS based malicious communication channels, therefore to characterize them based on the amount of the transferred data?

2. ***Question 2:*** How to quantify this abuse and identify the sources as well as the involved infrastructures?

3. ***Question 3:*** Even though the data is encrypted, is it feasible and possible to detect such malicious channels in DNS without any decryption or malware reverse engineering efforts?

4. ***Question 4:*** In terms of efficiency and scalability, is it possible to create a pDNS database to observe the changing behaviors of malicious payload distribution channels?

1.3 Contributions

The main contributions of this dissertation are:

Detection of payload distribution channels in DNS: A malicious payload distribution channel in DNS is a new concept, which is adapted by different malware families [29, 65]. Therefore, the analysis of these channels is very limited in the literature, and the proposed detection mechanisms are specific to certain malware families [29]. Our system is proposed to fill this gap in the existing research work. First, we present a thorough analysis of such channels, and discuss the techniques used by them. Based on the analysis, we model the behaviors of these channels to be able to categorize different C&C communication types. The novel detection

mechanism that we propose is based on the domain zone activities, which can detect even encrypted communication channels. Finally, we test our system with pDNS for one month, and we can detect long-lasting malware domains, which are previously unknown.

Design and implementation of a pDNS database: We use pDNS which is very useful for security investigation, however during our analysis on malicious payload distribution channels, we needed a historical pDNS database. We are required to have a scalable and efficient design, which can handle tremendous amount of the pDNS traffic. Also, the lookups to the database have to be efficient. We implemented an API and a web interface to access to the database. It is important to mention that after the completion of writing this dissertation, my colleagues in NCFTA has started working on the design of a pDNS database, which is based on a library (mtbl) [38] developed by Farsight Security, Inc. [1]

1.4 Structure

The rest of this dissertation is organized as follows. In Chapter 2, we detail the background information on DNS, DNS tunneling, and pDNS as well as existing related work that are relevant to this dissertation. Chapter 3 presents our work on the analysis and detection of malicious payload distribution channels in DNS. Afterwards, we present our pDNS database design and implementation in Chapter 4, and in Chapter 5 we conclude the work presented in this dissertation, and provide some future research directions.

Chapter 2

Background and Related Work

2.1 Background

In this section, we introduce some of the concepts that are fundamental background to the work presented in this dissertation. The section starts with a presentation of DNS, and later we highlight DNS tunneling. Afterwards, the pDNS technique is explained.

2.1.1 Domain Name System

The DNS protocol is designed to be a translation service for the Internet infrastructure. Every web request to a domain name is initiated by a DNS query to receive an IP address, which corresponds to the domain name. Hence, remembering domain names instead of numerical IP addresses simplifies the use of the Internet.

Fully Qualified Domain Name (FQDN) is a domain name that shows the location of a computer in the DNS hierarchy. The structure of a FQDN can be considered as a tree with multiple parts, which are placed according to a hierarchy. A FQDN `users.encs.concordia.ca.` has multiple labels, which are separated by dots. The rightmost label (ca.) is the highest level and the leftmost label (users.) is the lowest level in this hierarchy. The start of the hierarchy is the Top-Level Domain (TLD), therefore labels are named accordingly. `ca.` is the TLD, `concordia.ca.` is the Second-

Level Domain (2LD), `encs.concordia.ca.` is the Third-Level Domain (3LD), and so on. Any label that comes after the 2LD is considered as a sub-domain of the 2LD.

In DNS, response data is stored in authoritative name servers that are responsible for a particular domain name. These name servers are named as Start of Authority (SOA), and they hold the original information of a domain in a zone file. Although authoritative name servers are the center of information for their authoritative zone, they are often configured to be a master server, and slave name servers are setup to respond to DNS queries. This is an optimization in the network level by system administrators, especially for domains that receive a large amount of DNS queries, e.g., `google.com`. Therefore, a zone file is an important element of an authoritative zone in a DNS hierarchy. It defines the services running under a particular domain. The entries in the zone file are called RRs. In a zone file, a 2LD name usually has multiple RRs dedicated for different purposes (see Figure A.1 in Appendix). These records consist of five main components: name, class, type, Time to live (TTL), and data. The RR name is in FQDN form, and the sub-domains of 2LD are defined and mapped to the corresponding RRs in the zone file. In some cases, a wild-card (*) can be used as a label to return the same RR for any sub-domain [63]. A sub-domain can be setup to have its own zone file with a dedicated name server. In this case, the name server of the 2LD delegates queries to the name server of the sub-domain. This technique is called zone delegation, and it is often used for easing the management of different sub-zones under a domain [47]. The length of the RR name cannot exceed 256 bytes and each label length is limited to 63 bytes [64]. The RR class defines name spaces, that are used for different purposes within the DNS protocol. The default value for RR class is IN, which stands for the Internet. The RR

type indicates the type of information carried by the DNS message. In Table 2.1, we list some of the RR types used in our work. The TTL value is a time period used by DNS servers to determine how long to cache the response before discarding it. The RR data is the response information assigned to a RR name.

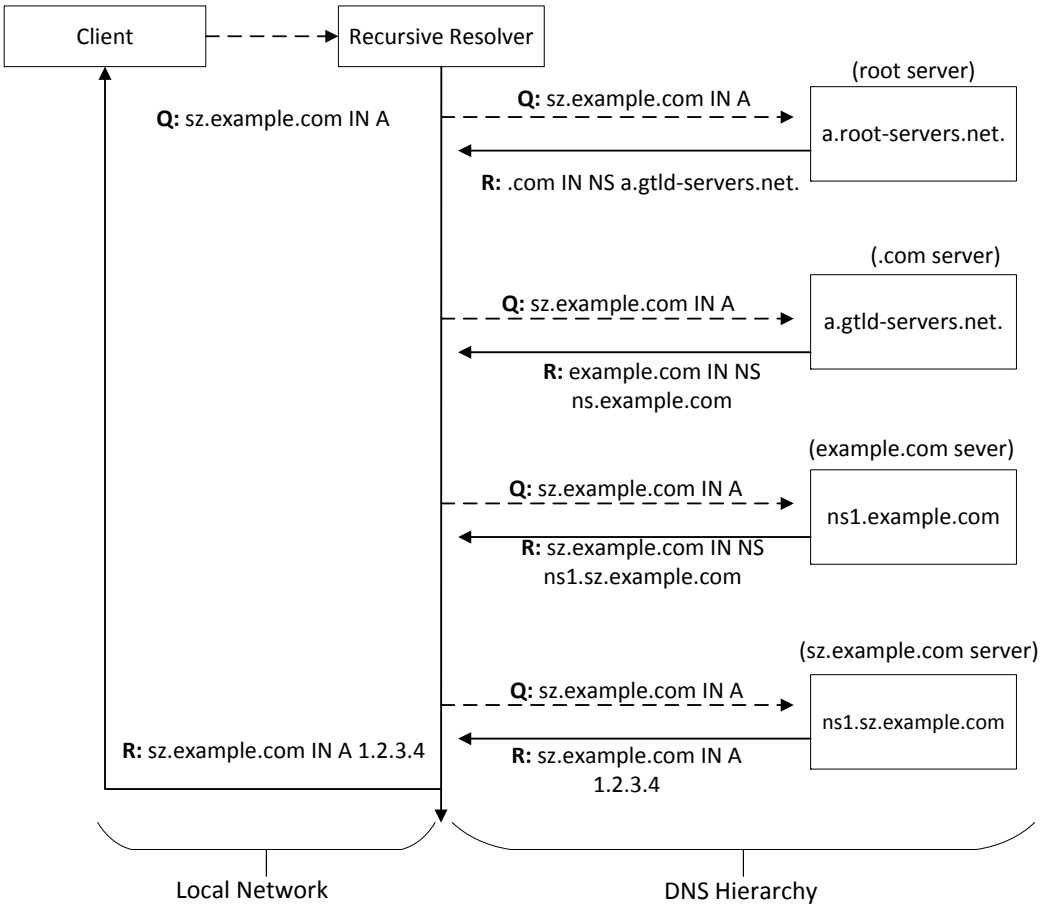


Figure 2.1: *Recursive DNS Query and DNS Hierarchy*

The protocol is designed to be a simple database lookup with queries and responses. The query is started at the host machine by its stub resolver, which is a simple resolver that initiates DNS queries. Stub resolvers delegates the query to a local DNS resolver, which is defined in the host settings. The DNS resolver then

RR type	Description
A/AAAA	IPv4/IPv6 address
NS	Name server
MX	Mail server
TXT	Text information associated with a name
CNAME	Canonical name or an alias name

Table 2.1: *DNS RRs Used in our Work*

interacts with the DNS hierarchy to receive the answer for the query. In a typical scenario, a user puts a FQDN (`sz.example.com`) in the address bar of a web browser to start browsing the Internet. As he presses the enter key, the stub resolver on the host machine sends a DNS query to the local DNS resolver as seen in Figure 2.1. If the resolver does not have a cached copy of the RR, it starts querying the DNS hierarchy by starting from the root DNS servers. These servers are the backbone of the entire DNS infrastructure and they hold the IP address information of authoritative name servers of TLDs. Based on the TLD of the query, root servers return the corresponding IP address of a name server to the DNS resolver. The next step, the DNS resolver queries to this name server to receive the IP address of the name server that is authoritative for the 2LD of the query. As seen in Figure 2.1, the recursive query continues until it reaches to the final zone, where the response of the original query is stored.

DNS-based Security Measures

Sender Policy Framework (SPF) [89], Domainkeys Identified Mail (DKIM) [5], Domain-based Message Authentication, Reporting and Conformance (DMARC) [55], and Opportunistic Encryption (OE) [78] are existing specifications, which are facilitated by Internet Engineering Task Force (IETF), that use DNS as a part of their mechanism.

They are used to protect domains, hence domain owners, against different types of security threats. To implement these specifications, the domain administrators create entries in the zone file of their domains. Because the operational data of these specifications is in text form, they are often stored in TXT RR, which is the a flexible RR type in terms of the syntax format.

SPF defines the authenticated mail servers that can use a particular domain for emails. When a mail server receives an email, it can verify whether the received email can be sent by that particular IP address or not by querying the TXT RR of the 2LD of the email address domain. The same applies to DKIM, which is used for authenticating emails that are sent by a domain. This security mechanism is rather applied to emails by verifying a signature in the email header. The verification is done by simply receiving the public key of the email address domain, which is stored in the TXT RR. DMARC is a specification that creates a layer between email recipients and existing specifications, i.e., SPF and DKIM. It is designed to simplify the handling of these specifications, and eventually to promote the use of them. There are also other known methods that store some data in TXT RRs such as OE, which uses DNS as a means to distribute public keys for different purposes [78].

DNS Tunneling

The concept of using DNS for data transfer is introduced by Dan Kaminsky [53]. He demonstrates the feasibility to use DNS queries to exfiltrate data, and DNS response packets to receive data. As seen Figure 2.2, tunneling queries and responses are long due to the embedded data. In the query, the data is added as a sub-domain label, and it might be multiple labels due to the restriction in the length of labels. The RR

type can be TXT, CNAME, or NULL. However, it is often TXT because CNAME is more limited in terms of syntax format, and NULL might be dropped by some DNS resolvers. Both outgoing and incoming data is encoded with Base64. However, there is no restriction in the encoding scheme as long as it uses a suitable alphabet for DNS [63].

DNS query:
Go6fdtgZ0llQyAx1VWJwvzdHdxMpHu6xfMRq8sVSfqwPvI9T.dnstunnel.com IN TXT

DNS response:
xMtwHYRyZuz4QbhBKZlVWvPBfiuGjb1WQxtZN7PR9Wf0sfnAqxDOJD9LgmwfFaU
Go6fdtgZ0llQyAx1VWJwvzdHdxMpHu6xfMRq8sVSfqwPvI9TEIV8pkXw4P4TCSH05
BAO1LGPMQXDTYLY2woxM1j06mCMhrNjWzI8WbmCBlj2dpR73KBnDlDRmheKWMJ
x2dUTp4iFMH4N9kXjeOYis==

Figure 2.2: *DNS Tunneling Query and Response*

DNS tunneling can be used to bypass restricted networks, such as commercial WiFi hotspots. These networks allow all DNS traffic. However, they require authentication to connect to the Internet. In this context, DNS is used as a carrier for other protocols by embedding outbound and inbound traffic into query and response messages respectively. The possibility of having free Internet connection grabs attentions, and then several open-source DNS tunneling tools are developed [61].

In a normal scenario, there are two key players in a DNS tunnel; a host and a remote server as seen in Figure 2.3. The server and the host can be any computer that has access to the Internet. Once these machines are ready, a DNS tunnel user has to have control over the zone file of a domain. A domain name is required to orchestrate the traffic, and it can easily be obtained for free by using dynamic DNS providers (e.g., http://freedns.afraid.org). The user has to set an NS record, which

points to his remote server (`ns.dnstunnel.com`). Therefore, any DNS query to that domain will reach to the name server of that domain. The remaining part of the setup is that the user has to set up one of the existing DNS tunneling tools on both the host and the remote server. The host script will encode the data and create DNS queries to the domain in question (`dnstunnel.com`). The remote server decodes the data, and replies with the demanded response. In Figure 2.3, the remote server plays a proxy role between the user and the Internet.

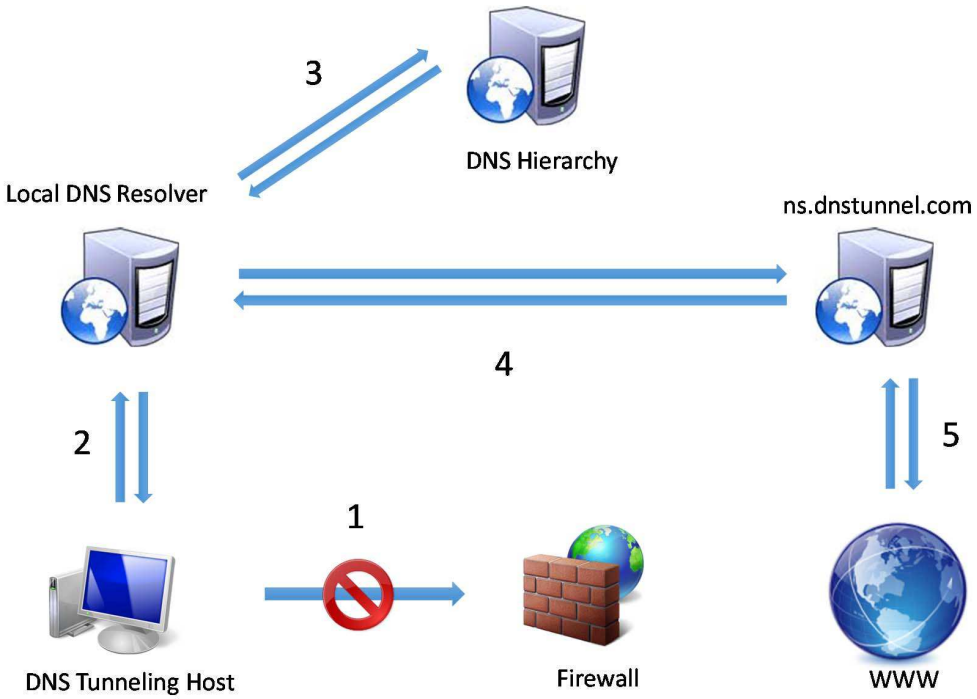


Figure 2.3: *DNS Tunneling*

The feasibility of using DNS RRs to distribute payload has been proven by the DNS tunneling technique, which shows that DNS can be used for transmitting any type of information after simple encoding operations. However, there are some limitations due to the low data transmission rate through RRs. DNS response packets

are limited to 512 bytes if Extension Mechanisms for DNS (EDNS) is not used [85]. EDNS is an extension mechanism for DNS to enhance the protocol based on the increasing capacity of network systems, which increases the DNS packet size up to 4096 bytes. However, some firewalls might not accept DNS responses that are larger than the default 512 bytes. This potential drawback forces botmasters to find stable ways to utilize DNS for payload distribution channels.

2.1.2 Passive DNS

Passive DNS is a technique to replicate the global DNS activities in order to investigate it in near real-time. Florian Weimer introduces the first pDNS data collection mechanism [87]. In his proposal, the initial aim is to fix the inconsistency between A and PTR records. PTR records are used for reversed lookup to find out the domain name of a given IP address. Therefore, the mappings between IP addresses and domain names require constant updating due to dynamic IP addresses. Therefore, DNS can be replicated, and it can be used as a historical database to run such reversed queries as well as to gain other intelligence on domain names. There are several implementations of such pDNS replicating systems including the Security Information Exchange (SIE) initiative provided by Farsight Security, Inc. [1]

The Security Information Exchange (SIE) Initiative

Passive DNS in SIE is generally believed to be the most established implementation of Weimer's proposal. It has a distributed architecture to deploy replicating sensors all around the globe, and build a central information center [26, 30]. The system is fed by globally deployed sensors, which are placed in or near recursive name servers

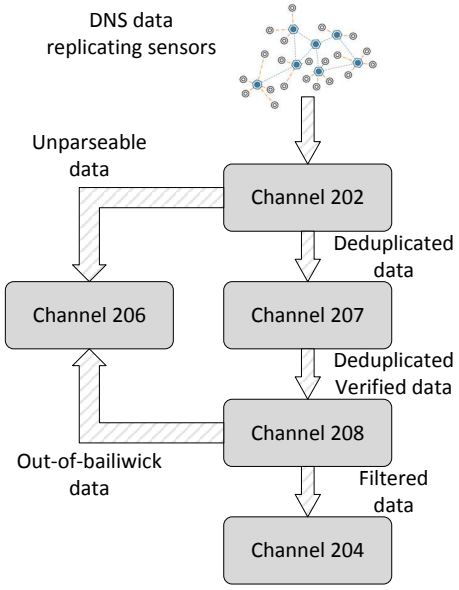


Figure 2.4: *Passive DNS Channels in SIE*

in different networks. These sensors duplicate the traffic passing through these name servers, and upload the data to a central processing pipeline.

As seen in Figure 2.4, pDNS replicating sensors upload captured query and response DNS packets to the infrastructure of SIE. These channels are actually Virtual Local Area Network (VLAN) connections, which can be accessed for different purposes. Initial uploads from sensors are sent to the channel 202, which outputs the raw data. If a packet is missing response, or there is only response; it is discarded, and sent to channel 206. From channel 202 to 204, repeating RRs are combined into a single RR by de-duplication and re-de-duplication processes. Also, some filtering and blacklisting are applied to remove the artifacts of cache poisoning attacks [30].

2.2 Related Work

In this section, we first discuss existing works on the detection and mitigation of DNS abuses in two folds: protocol-level and system-level. Afterwards, we focus on recent research efforts in terms of establishing and analyzing pDNS to detect existing and emerging security threats.

2.2.1 DNS Abuses

DNS is often targeted by malicious networks for different purposes. It can be either hijacking a victim's web request, or establishing a resilient network with multiple layers of proxies. In any case, DNS is abused by botmasters to accomplish their malicious goals. Based on the nature of abuses, they can be categorized as protocol-level and system-level as seen in Figure 2.5. In the following sections, we will present an overview of the research works on detection and mitigation of DNS abuses at different levels.

Protocol-level Abuses

The naive architecture of the DNS protocol allows botmasters to use it to establish covert communication channels or fast-flux networks by simple tweaks within the protocol. These abuses show that the DNS protocol allows attackers to transfer information in DNS query and response packets [29, 65, 76] or to create malicious networks by pointing thousands of compromised machines with a single domain [20, 49, 66].

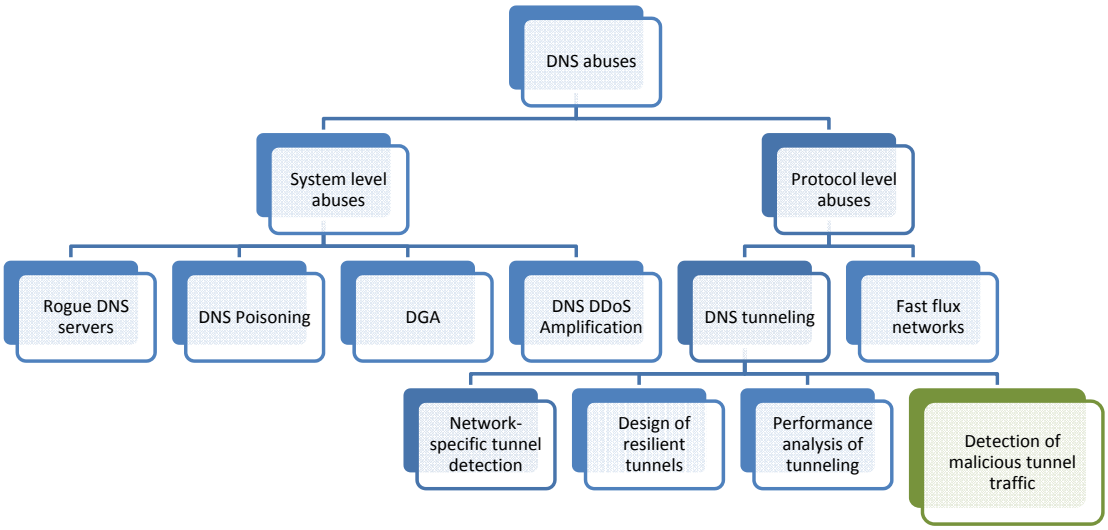


Figure 2.5: *The Placement of Our Work in the Literature*

DNS Tunneling: The use of DNS as a communication medium for payload distribution is relatively new and research activities on this topic are limited. Although, these studies are scattered, they can be roughly grouped under two categories: malicious attack payload distribution channels in the DNS protocol, and the DNS tunneling.

Dietrich et al. [29] first discuss the existence of botnets that tunnel C&C communication channels through DNS. They discovered a malware family, which is named Feederbot, that exfiltrates data within DNS query sub-domain labels, and infiltrates attack payloads in DNS response packets. Their detection method introduces extraction of several features from the response data. While their work shows promising results, it is limited to the detection of aggressive DNS tunnels for C&C channels. Some malware families use more resilient methods for receiving attack payloads through DNS rather than the DNS tunneling [65]. Also, their work focuses on the assumption that there will be a certain degree of traffic, while our analysis shows that

some families use DNS to receive a very limited amount of payload, e.g., the Morto malware family [65]. Moreover, we find that malware might not receive Base32 or Base64 encoded payload, rather clear text in TXT records. Xu et al. [19, 3] introduce a resilient mechanism for bots to create covert channels through DNS for C&C communications. They design a stealthy C&C architecture that supports two different modes. The *Codeword* mode creates a uni-directional communication channel that pulls the attack payload. The *tunneled* mode creates a bi-directional communication channel between bots and the C&C server. They also mention some techniques to increase the stealthiness of these channels to make them virtually undetectable from the compromised host's perspective. In fact, during our analysis in pDNS and malware datasets, we find that their proposed methods are already used by some malware families such as Feederbot, Morto [29, 65]. While their technique can easily defeat the host-based detection mechanisms, we are able to detect these malicious channels in the pDNS traffic. We also found that malware families, which use bi-directional channels, are often detected easily due to their extensive traffic. Similarly, Raman et al. [76] propose a network penetration technique that uses the DNS tunneling to infiltrate the attack payload. Their method is based on establishing a tunnel by an exploit code. Our system can detect the payload distribution channel in pDNS regardless of the format of the payload, as we do not inspect the content of DNS messages. Also, there is an ongoing effort from IETF in preventing name servers being abused by botmasters for attack payload distribution [44].

DNS tunneling has gained a growing interest in the academia, as it offers a wide-range of opportunities to establish covert channels in DNS. So far, it has been studied from different perspectives, such as security, feasibility and performance. Dušan

Bernát [11] formalizes the use of DNS as a communication medium by modeling a storage-like read/write mechanism in the protocol. The studies in DNS tunneling also focus on the detection of such covert channels in network traffic. Especially after development of several DNS tunneling tools, researchers have emerged to design detection techniques for the traffic of these tools. There are some proposed methods for detecting DNS tunneling within a network by using n-gram analysis [16, 17, 75]. They present promising results in terms of detecting the tunnels. However, malicious payload distribution channels often do not have extensive upstream data; thus they do not show this characteristic feature of DNS tunneling tools. Therefore, any string based analysis on the queries might not reveal enough differences between regular and malicious queries to detect these channels. Also, our system detects payload distribution channels regardless of the syntax by using DNS zone activities (Section 3.3.3).

Greg Farnham [33] discusses existing open-source DNS tunneling implementations, and proposes a mechanism for the detection of these tools. This mechanism is mainly based on using a set of rules for Intrusion Detection Systems (IDSs). Although IDSs are effective for capturing malicious data streams in network traffic, signature-based systems are often prone to false positives [81]. Several studies [4, 61, 84] analyze existing DNS tunneling tools from the performance perspective, as DNS tunneling often raises concerns on the limitation of DNS packet sizes. They conclude that these tools create a significant overhead in the network traffic . Finally, Ellens et al. [31] apply network flow analysis methods on DNS tunnels to detect them accurately within the network boundaries. It shows that DNS tunnels can be investigated like any other network-based covert channel. However, it is important to mention that this work and the proposals reported in [16, 17, 75] are limited to the detection of tunnels within

a given network, whereas we detect such channels in the global DNS traffic without limiting it to a single network.

Kenton Born [14] investigates the possibility of establishing covert channels in DNS by using web browsers. He shows that such channels can easily be initiated with web browsers' privileges, and any type of data can be exfiltrated without alerting IDSs. The author also investigates [15] the possibility of piggy-backing existing DNS packets without altering the packet structure. It is an alternative DNS tunnel as opposed to traditional DNS tunneling implementations [61]. There are several studies on building a covert channel by manipulating DNS query and response packets [18, 67]. They discuss sending and receiving data through DNS query and response packets in the traditional DNS tunneling paradigm (Section 2.1.1). Paxson et al. [69] propose a comprehensive solution for the detection of covert channels in DNS. The proposed solution is based on the possibility of the existence of different types of covert channels in DNS. For example, bots can exfiltrate boolean data by using predetermined values in DNS query sub-domains, or by timing the queries in a predetermined pattern. While the study shows promising results in live and pDNS datasets, it can be a complementary solution to our proposal as our work does not rely on the packet content.

Fast-flux Networks: Fast-flux networks are established by returning a different set of IP addresses for each DNS query, therefore same domain name is mapped to an extensive number of IP addresses. This approach makes it difficult to detect the machine behind that domain name. Botnets are increasingly adapting malicious flux networks because it gives them the freedom of establishing a protective layer in front of C&C servers [80]. The detection of these servers has become more complex, and

even if they are detected, most of the time botmasters reach their goals before domain take-downs. There are several studies [20, 49, 66, 68, 71] in the field of analysis and detection of malicious flux networks. Their methods are based on characteristics of these networks, and they use certain features, e.g., the diversity of IP addresses and name server records. Perdisci et al. [70] analyze fast-flux networks in pDNS, and as opposed to the previously proposed methods [49, 66, 68], they focus on the detection of such networks without limiting it to domains that are found in spam emails.

System-level Abuses

DNS is a simple yet powerful system that consists of globally deployed name servers. Every transaction in the Internet is initiated with a DNS lookup, therefore botmasters try to exploit every possible point in the whole system. Recent studies show that there are still unknown vulnerabilities almost at every layer of the system. The authors of [6, 24, 25, 52] study cache manipulation attacks on caching DNS in the DNS hierarchy. Antonakakis et al. [6] propose a system, namely Anax, for scanning caching resolvers to detect possible cache poisoning attacks. This system is built on top of the system, which is introduced in [24]. As opposed to this work, Jiang et al. [52] introduce an uncovered vulnerability in DNS, which allows botmasters to keep previously deleted domains alive. Although this vulnerability falls in cache manipulation type of abuses, it highlights a serious architectural issue in the current implementation of DNS. According to the attack definition, an attacker can get a domain resolved by an authoritative name server, which the attacker controls, and keep the cache alive in targeted open resolvers.

Dagon et al. [25] propose a novel method to resist against cache poisoning by forcing caching resolvers to use mixed case encoding. Therefore, an attacker needs to guess the encoding to find the cached data, and update it with malicious response data. Even though the method seems to be efficient, the length of domain names play an important role in terms of entropy. Domains with short names are likely to be guessed easier than domains with longer names. Moreover, domain names, which are composed of only numbers (e.g., `123.com`), cannot be encoded with mixed cases. Dagon et al. [27] investigate rogue resolvers, which resolve domains to malicious IP addresses. Compared to cache poisoning attacks, this type of attacks target stub resolvers rather than caching resolvers. In other words, DNS settings on a host machine are altered to use rogue resolvers to delegate DNS queries. Therefore, these resolvers map legitimate domains to phishing campaigns.

Studies on DNS based botnet detection show that newly registered malicious domains have different characteristics compared to a regular domain [7, 12, 21, 46]. For instance, newly registered malicious domains tend to resolve within a certain IP range, which is often in the control of botmasters [46]. One study [12] shows that the lifetime of a malicious domain is very short, because botmasters often use certain domains for dedicated campaigns. In [8, 12], the authors introduce techniques, which are deployable to the top level name servers, to detect malicious domains in the backbone layer of DNS. However, the positioning of the detection system is weak in terms of correlating attacks through multiple networks. Antonakakis et al. [7] observe domains for malicious activities in authoritative name servers. While this method remains more fine-grained compared to [8, 12], the detection mechanism can only detect domains that have similarity to previously seen domains. Choi et al. [21]

propose an algorithm to detect the botnet activities based on DNS queries. They target the similarity of queries of bots from the same botnet. Although they are focusing on query similarities, our work focuses on query and response patterns as well as the DNS zone activities.

Another emerging system-level abuse of DNS is Domain Generation Algorithm (DGA) based domains, which are randomly generated and queried by bots with the hope of finding the registered domain. Botmasters also generate the same set of domains concurrently, and register one of them. In this way, law enforcements and other security organizations have difficulty to find malicious domains among thousands of newly generated domains per day. This approach is also known as domain fluxing [83]. There are several studies [9, 83, 90] introduce methods for the detection of botnets, which use domain fluxing. Stone-Gross et al. [83] report the existence of such malicious technique, specifically by a malware family Torpig. Interestingly, Torpig uses Twitter as the source of randomness for generating random domain names. On the other hand, [9, 90] consider the fact that bots receive a large amount of NXDOMAIN (Non-Existent Domain) responses, because randomly generated domains are not registered most of the time. NXDOMAIN response is returned when the domain in question does not exist. Yadav et al. [90] test their method with an offline datasets, whereas [9] deploy their system, namely Pleiades, in live traffic from two major Internet Service Providers (ISPs). Both systems show promising results with high accuracy in the detection of such botnets. However, [9] do not rely on any blacklists or predefined structure from any botnet, therefore it could detect previously unknown botnets.

2.2.2 Passive DNS Analysis

Passive DNS is definitely important for DNS related studies. So far, the number of studies, which use pDNS as a data source, has been increasing, and it seems that pDNS will become one of the de-facto data sources for threat analysis.

There are studies in using pDNS for detecting malicious activities in near real time [23, 26, 28, 58, 59, 60, 92]. All of these studies discuss the effectiveness of utilizing replicated DNS traffic for understanding the attack methodologies used by botmasters. Marchal et al. [59] introduce a framework, namely DNSSM, to build domain-based analysis by using DNS responses. From data capturing perspective, it uses Weimer's technique [87] as explained in Section 2.1.2. The system mirrors the live DNS traffic by capturing query and response packets in a recursive resolver. On contrary to the common practice, the authors designed their framework to store domain-based features, e.g., number of IP addresses, and TTL values. In common practice, DNS data would be stored as it is captured, and it would allow researchers to use the captured data for different types of analysis. In our pDNS database (Chapter 4), we rather follow the common approach. The authors use ten features to detect malicious activities, including a feature to pinpoint approximate geographic location of physical machines. Marchal et al. [60] design an architecture for the actual implementation of the pDNS analysis framework [59], that they have introduced. Because analyzing such intense data source requires distributed storage and computation, the authors utilize Apache Hadoop[1] and Apache Cassandra[2]. As a proof of concept, they test their framework and architecture with a sample data set. It shows that their set

[1]http://hadoop.apache.org/
[2]http://cassandra.apache.org/

of features, which were introduced in [59], are accurate in detection of certain types of DNS anomalies.

Marcha et al. [58] propose an approach to analyze pDNS in a semantic-aware manner. The approach involves using natural language processing methods to determine the potential of a domain to be malicious. The authors ascertain models of malicious domain names by determining potential keywords, which are used in phishing domain names. Phishing domains are often composed with eye-catching keywords, such as, pharmacy, paypal, and ebay. It is also important to mention that the system is scalable, even with the pDNS data from big ISPs.

Paul Vixie and Jun Murai [86] discuss the design and implementation process of one of the biggest pDNS deployments, SIE from Farsight Security, Inc. [1] Because SIE has deployed sensors in or near recursive name servers in the US and Europe, it requires a scalable architecture. One of the key elements of the architecture is to provide different channel outputs, which allow researchers easily investigate various security problem without putting extra effort for data processing. For example, one channel allows to analyze the raw DNS data, which comes right from the name servers, and another channel provides data, which have been sanitized from cache poisoning attacks. Similarly, Luciana Costa and Roberta D'Amico [23] introduce an architecture for the pDNS replication. However, their solution is not distributed, rather part of a security intelligence project for a local ISP. Zdrnja et al. [92] discuss building a security intelligence platform by monitoring DNS passively. Although their approach is similar to previous solutions, their evaluation is limited to a university network. From scalability perspective, their approach remains rather naive because it is limited to a network, and requires improvement in terms of the scalability.

Finally, Deri et al. [28] introduce a unique method to the pDNS analysis. Their system observes domains for existing and emerging economical trends within the authoritative zone of the Country Code Top-Level Domain (ccTLD) registrar of Italy, which is responsible of `.it` TLD. The system is deployed in the name servers of this registrar, and logs the DNS traffic passing through these servers. These logs are analyzed to detect trends in domain names, and the results of the analysis are reported to a central data store. However, the system lacks the intelligence on Italian domains, which use other TLD, i.e., `.com`, `.org`. Also, the work is limited to the analysis of domain names, and it requires a semantic-aware analysis of the actual content of the websites.

Chapter 3

Malicious Payload Distribution Channels in DNS

3.1 Introduction

A common approach to bypass network defense borders is by tunneling the communication through existing protocols. Such tunneling can effectively defeat traditional firewalls and IDSs. Botmasters also often prefer tunneling to keep their communications under the radar. In the early stages of botnets, botmasters mostly used IRC channels (e.g., Agobot [48]) to operate and control their activities. The advancement of newer protocols (e.g., instant messaging, P2P, and HTTP) largely outdated the use of IRC channels [2]; see e.g., Zeus [32] (HTTP based), Storm [50] (P2P based). As a natural extension to exploiting common protocols for tunneling, DNS comes into play due to its wide availability. DNS is a query and response protocol, which responds to each query with the corresponding pre-defined RR. The simple but robust architecture of DNS attracts botnets to abuse the system for different malicious activities [9, 25, 27, 29].

In 2004, Dan Kaminsky [53] demonstrated the feasibility to bypass restricted networks that allow all DNS traffic, such as commercial WiFi hotspots, that require authentication to connect to the Internet. In this context, DNS is used as a carrier for other protocols by embedding outbound and inbound traffic into query and response

messages respectively. Since then, DNS tunneling has been used to design several application tools [61], which operate covert channels through the public DNS infrastructure. Moreover, these tunnels can be established by using free DNS providers, which are already known to be abused for different types of malicious activities [10]. In RSA 2012, Skoudis [82] mentions an information theft case, which is carried out by a malware family using the DNS protocol to exfiltrate information.

Botmasters take advantage of DNS tunneling to conduct malicious activities such as C&C or payload distribution. In payload distribution channels, for instance, botmasters use DNS query and response packets to carry out malicious instructions and payload updates to individual bots. Recently, a few malware families have been identified as using the DNS protocol to hide their communications, including Morto [65], Katusha [2], and Feederbot [29].

Due to the inherent nature of DNS, it is quite inefficient as a payload distribution channel compared to other protocols, which botmasters often use [84]. However, DNS infrastructures still have been abused by a few botnet families in the recent past. Such examples indicate that botmasters are willing to exploit DNS as an attack channel due to its wide availability. Previous works on DNS abuses [29] mainly focused on specific botnets, and DNS abuses have not been comprehensively studied as compared to e.g., P2P botnets [50].

In this work, we propose a detection mechanism for DNS payload distribution channels by leveraging some inherent features of DNS as used by malicious and non-malicious domains. We use this mechanism to analyze a significant amount of DNS traffic to understand the extent of DNS abuses in the wild. We have detected few previously unknown long-lasting malware domains and different types of payload

distribution channels. Considering the fact that DNS is vulnerable to such attacks, our system puts the defense line one step ahead of the botmasters. Moreover, our proposed technique, which is based on the analysis of RRs, shows promising results regardless of syntax format of payload distribution channels.

The main contributions of this work are as follows:

- **Thorough analysis of malicious payload distribution channels:** We present an analysis of these channels with 1-year malware dataset covering Jan.-Dec. 2012.

- **Characterization of DNS messages:** We find that malicious networks used different techniques for distributing attack payloads, including an indexed query pattern to distribute the attack payloads in multiple parts. We introduce a technique to determine channel patterns, and discuss the feasibility of each pattern. We find that most of the malware instances are using a resilient pattern to retrieve the attack payloads.

- **Detection of payload distribution channels using the pDNS traffic:** We propose a method to detect payload distribution channels based on the analysis of resource record activities, and determine the intensity of the distribution by using a fuzzy set theory.

- **Evaluation of the system in the pDNS traffic:** We experiment our system on 3 different datasets: pDNS, pDNS database, and a dataset of 1-year malware dynamic analysis reports.

3.2 Background

In this section, we outline key differences between DNS tunneling and payload distribution channels. Then, we discuss the use of these channels both for legitimate and malicious purposes.

3.2.1 Payload Distribution via the DNS Hierarchy

Recently, DNS has become a target to distribute malicious payloads for two main reasons. First, DNS traffic is often allowed to pass without inspection in corporate networks, as it is considered to be a core element of the Internet activities. Second, the DNS protocol has some fields that are defined to be more flexible, which opens the doors for other unintended uses. Malicious payload can be stored in different RRs (e.g., NULL, TXT, or CNAME). The payload data can be cached in DNS resolvers, and they can be accessed even if C&C servers are down. Also, the labels within the RRs name can be used to store Base32 encoded data. Request for Comments (RFC) 1464 paves the way for payload distribution by opening the possibility of storing arbitrary information within DNS messages [79]. However, it recommends to store key-value pairs to share some operational data between servers. The feasibility of using DNS RRs to distribute payload has been proven by the DNS tunneling technique, which shows that DNS can be used for transmitting any type of information after simple encoding operations. However, botmasters face some limitations due to the low data transmission rate through RRs. In general, payload distribution channels are established in the same way as DNS tunnels are established [84].

3.2.2 Use Cases of Payload Distribution

Payload distribution through DNS is relatively a new concept, and it has very limited number of legitimate uses. Some organizations have been inspired by the evolution of DNS tunneling, and started to use DNS as a means to channel a part of their operational data to enhance their systems.

Legitimate Use Cases: In 2007, Trend-Micro Inc. proposed a method to distribute malicious code signature updates through the DNS protocol [57]. The intention of this technique is to feed anti-virus client software with signature updates through DNS, as an alternative update mechanism. The signature updates are divided into several chunks, which can be identified by an identifier number. These pieces are encoded with Base64, and assigned to the zone file as TXT RRs of a specific domain name. When the client needs an update of a malicious code signature, it sends a query with an identifier number of the signature as the FQDN label. Then, the server responds with the corresponding anti-virus signature in TXT records. In general, each signature update can span over many TXT records, which makes the client generate many queries to retrieve the whole pieces of the update. Finally, the client combines all TXT records, and then forms the actual update of the malicious code signature.

In 2009, Devicescape Software Inc. introduced a system for public hotspot authentication systems for mobile devices [45]. In their model, there are public WiFi hotspots, which are placed across many places such as coffee shops, and restaurants. The authentication system for these hotspots is managed through a centralized scheme. The DNS protocol is used as a channel to transfer authentication parameters

between mobile devices and a credential server. The client software prepares a DNS query, which consists of six sub-domain labels to carry different parameters, e.g., the Media Access Control (MAC) address of the client's machine. When the name server receives the query, it forwards the parameters embedded in these labels to the back-end credential server. Based on these parameters, the credential server prepares the corresponding authentication response to be transported back to the client. Finally, the client verifies the response, and then submits it to the authentication server in the local hotspot network.

Malicious Use Cases: The crucial component of any malicious network is the communication method, which should be resilient and efficient. Recently, DNS has been used by malicious networks for updating bots with recent payload data (i.e., module updates, command instruction). In 2011, Dietrich et al. [29] reverse engineered the Feederbot botnet, which uses DNS as a C&C channel. Another example of the abuse of the DNS protocol is the Morto worm, which uses DNS TXT records to transmit a single piece of information. The embedded information is a Uniform Resource locator (URL) that points to the real attack payload, as explained by Symantec in [65].

3.3 System Description

3.3.1 Overview

Our system monitors DNS queries and responses in pDNS, and detects payload distribution channels established within DNS messages. As shown in Figure 3.1, the system consists of two main modules: query and response pattern and payload distribution detection modules. Initially, the system divides the captured DNS traffic stream into

epochs $E = \{e_1, e_2, \cdots, e_m\}$ (e.g. $epoch = 1$ day). For each epoch, it aggregates the DNS queries and responses of a given domain name d. These collected messages are sent to the query and response pattern analysis module, which determines the channel pattern. Then, the collected messages are sent to the payload distribution detection module which pulls all the DNS RR activities of the domain from the pDNS database. This module finally determines the intensity of the payload distribution based on the zone activities. We also introduce two filtering mechanisms because some legitimate domains might resemble payload distribution channels. In the following two sections, we show how these modules interact to characterize, and detect payload distribution channels established in the pDNS traffic.

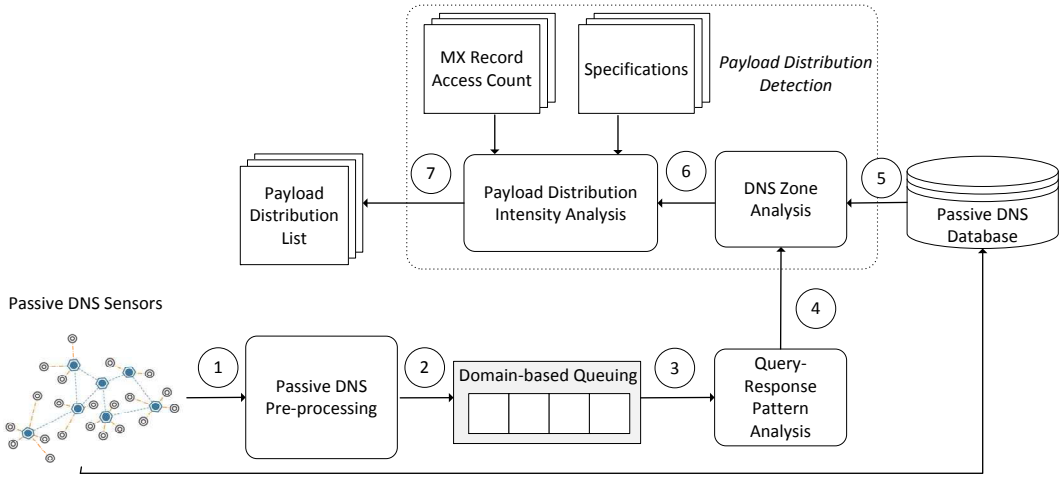


Figure 3.1: *System Overview*

3.3.2 Query and Response Patterns

The DNS protocol is based on query and response messages, which are used to find IP addresses of domains. A query from any client can be formed to retrieve different information from a name server, which will respond accordingly. By observing the

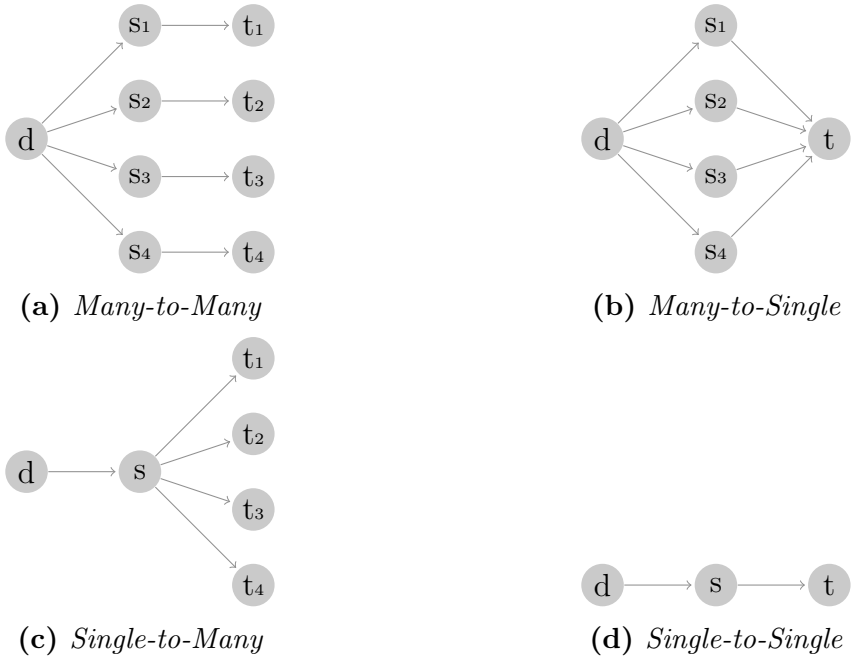
communication between client and server, we can model the relation between query and response messages. The query and response relations can be used to distinguish between different behaviors of payload distribution channels. When we observe any payload distribution activity, we have three parameters which are used to establish the channels in DNS. These parameters are the 2LD, sub-domain, and TXT record. The 2LD is the domain name, which is used to orchestrate the payload distribution activity. The sub-domain is used to transfer any information from a client to a server. The TXT record is the response information from the server to the client. During any session, the client and the server agree on a specific domain name to be used for the payload distribution activity. So, the 2LD parameter is determined before any session. Now, we are left with two parameters that are used to form the communication channel. Based on the nature of the payload distribution channel, these two parameters have different behaviors. The aim of query and response pattern analysis module is to differentiate between different behaviors of payload distribution. To achieve this goal, we analyze the exchange behavior of query and response messages. This module is built based on two observations:

Observation 1 *Payload distribution channels through DNS are forced to transfer small quantities of information with each DNS message, because DNS response packets are limited to 512 bytes of characters if EDNS is not used (see Section 2.1.1).*

Observation 2 *When transferring more information through DNS protocol, it results a significant amount of DNS queries and responses between the client and the name server (see Section 2.1.1).*

Figure 3.2 shows four possible payload distribution scenarios, which are based on the relation between sub-domains and TXT records. Figure 3.2a explains how

the client changes the sub-domains to send data to the name server, which responds with the corresponding TXT records for each sub-domain (Many-to-Many relation). Figure 3.2b shows how the client changes the sub-domains to update the name server about its status, and the server replies with the same TXT record for all possible sub-domains (Many-to-Single relation). Figure 3.2c explains how the client sends the same sub-domain that is answered with several TXT records from the server (Single-to-Many relation). This case rarely occurs within a small period of time, because these responses are stored by caching resolvers for a period of time (see Section 2.1.1). Figure 3.2d shows how the client sends the same sub-domain, which is answered by only one TXT record from the server (Single-to-Single relation).



(a) *Many-to-Many* **(b)** *Many-to-Single*

(c) *Single-to-Many* **(d)** *Single-to-Single*

Figure 3.2: *Query and Response Exchange Patterns (d:2LD, s:sub-domain, t:TXT records)*

Definition 1 *The query and response pattern model is a tuple* $G = \langle (D \cup T \cup S), E \rangle$, *where:*

- $D = \{d_1, d_2, \cdots, d_n\}$ *is a finite set of domain name nodes,*
- $S = \{s_1, s_2, \cdots, s_m\}$ *is a finite set of sub-domain nodes,*
- $T = \{t_1, t_2, \cdots, t_k\}$ *is a finite set of TXT record nodes,*
- $E \subseteq (D \times S) \cup (S \times T)$ *is a finite set of pairs of distinct nodes, called edges.*

We model the query and response relationship for each domain using a directed graph as captured by Definition 1. For each vertex v in G, we define two functions: the in-degree of v, which is denoted by $inD(v)$, returns the number of *entering* edges to the node v: $inD(v) = |\{u \in V \mid (u, v) \in E\}|$, and the out-degree of v, which is denoted by $outD(v)$, returns the number of *leaving* edges from the node v: $outD(v) = |\{u \in V \mid (v, u) \in E\}|$.

As shown in Figure 3.2, the query and response patterns share some properties as given by Property 1.

Property 1 $inD(D) = 0$, $outD(T) = 0$, $outD(D) = inD(S)$, $outD(S) = inD(T)$

In order to distinguish between the patterns shown in Figure 3.2, we determine the distance between two integer values, which might be cardinalities of any given sets, i_1, i_2 by using a normalized distance function as formulated in Equation 3.1.

$$Dis(i_1, i_2) = \frac{|i_1 - i_2|}{max(i_1, i_2)} \tag{3.1}$$

Since the query and response patterns can form complex relationships, we extract the commonly used pattern. When we compare between the degree values of two nodes, we extract the strong *node* candidate from each targeted set. In our case, we select the node, which has the largest degree value, since it reflects the common pattern behavior. Each query and response pattern can be recognized by the following properties:

Property 2 *Given that* $t \in T$ *and* $inD(t)$ *is the largest value in* $inD(T)$*, then Many-to-Many pattern holds when* $outD(D)$ *is not close to* $inD(t)$ *and* $|S|$ *is equal to* $|T|$*.*

Property 3 *Given that* $t \in T$ *and* $inD(t)$ *is the largest value in* $inD(T)$*, then Many-to-Single pattern holds when* $outD(D)$ *is close to* $inD(t)$ *and* $|S|$ *is equal to* $|T|$*.*

Property 4 *Given that* $t \in T$ *and* $inD(t)$ *is the largest value in* $inD(T)$ *and* $s \in S$ *and* $outD(s)$ *is the largest value in* $outD(S)$ *then Single-to-Many pattern holds when,* $outD(s)$ *is not close to* $inD(t)$ *and* $|S|$ *is equal to* $|T|$*.*

Property 5 *Given that* $t \in T$ *and* $s \in S$*, then Single-to-Single pattern holds when,* $outD(s)$ *is close to* $inD(t)$ *and* $|S|$ *is equal to* $|T|$*.*

Algorithm 1 shows an overview of the query and response pattern recognition in four steps. Step 1 (Line 1 in the algorithm) is taking a snapshot from the pDNS channel for a pre-defined window of time. This step produces a set of query and response messages for each domain that appears within the targeted window. Step 2 (Line 3 in the algorithm) is processing every domain name by constructing the relation graph between sub-domains and TXT records. Step 3 (Lines 4-8 in the algorithm) is calculating the out-degree vector for all sub-domains, in-degree vector

for all TXT records, and the out-degree of the domain. From these vectors, we get the largest degree, which is considered as a strong representative for the relation between sub-domains and TXT records. Step 4 (Lines 9-10 in the algorithm) counts the distinct values of sub-domains and TXT records. Step 5 (Lines 12-23 in the algorithm) determines the pattern mode based on the properties of each pattern. The order of the properties in if statements are arbitrary.

Algorithm 1: ExtractQueryResponsePattern

Input: A domain name d, set of sub-domains $S = \langle s_1, s_2, \ldots, s_n \rangle$, set of TXT records $T = \langle t_1, t_2, \ldots, t_m \rangle$

Output: Query and Response pattern mode, $\{Many\_Many, Many\_Single, Single\_Single\}$

```
1  D ← getSnapshoptFrom_pDNS(w)
2  foreach Domain d do
3      G ← Create_Relation_Graph(d, S, T)
4      SubDomain_Degree ← Max(outD(S))
5      TXT_Degree ← Max(inD(T))
6      Domain_Degree ← Max(outD(D))
7      SubDomains_Counter ← |S|
8      TXT_Counter ← |T|
9      Pattern_Mode = None
10     if Property 5 then
11         Pattern_Mode = Single_Single
12     else
13         if Property 2 then
14             Pattern_Mode = Many_Many
15         else
16             if Property 3 then
17                 Pattern_Mode = Many_Single
18             else
19                 if Property 4 then
20                     Pattern_Mode = Single_Many
21     return Pattern_Mode
```

3.3.3 Payload Distribution Detection

DNS Zone Analysis

Name servers play the main role during the lifetime of any DNS query. These servers are capable of handling any DNS query and returning the corresponding responses, which are taken from a zone file. In Section 3.3.2, we have seen four different methods, that are used to distribute data through DNS. Because name servers are key players in DNS, malicious networks need to have access to a name server for managing the payload distribution. After the name server is configured to be authoritative for the malicious domain name, botmasters prepare the zone file of the domain to hold all attack payloads for the delivery through DNS.

In DNS zone analysis module, we analyze the behavior of domain names by observing DNS zone files. Within the zone file of each domain name, there are different types of RRs. Each RR indicates specific services or operations associated with the domain name. One of the powerful features of the pDNS database is the aggregation of how many times each record has been requested, called access count (ac). In general, domain names, which are solely used for payload distribution, show different behavior compared to regular domains. Regular domains receive queries for different RRs. On the other hand, malicious domain names, which are only used for payload distribution through DNS, are only accessed to receive attack payloads. Therefore, they only focus on using specific RRs that are known to be used in payload distribution channels such as TXT records. Moreover, these domains do not heavily use the RRs that are normally used by regular domain names, such as A, AAAA, and MX

resource records. By observing the RRs and their access counts, we can profile the DNS zone activities of a domain name.

Extraction of DNS Zones: In payload distribution channels through DNS, name servers are considered as the payload distributors. Since domain names can have multiple zones, we must recognize the responsible zones, which are associated with payload distribution. This process can be formalized as an ordered set of labels L of a 2LD D within a period of time t, where the leftmost label represents the lowest zone within the DNS hierarchy. $D_t = \{L_1, L_2, \ldots, L_m\}$, $L_i = \{label_1, label_2, \ldots, label_{n-1}, label_n\}$, where $1 \leq i \leq n$, and m is the number of query and response packets that are captured under that 2LD, and n is the number of labels in each query.

A query might have multiple sub-domain labels, which might point to sub-zones under the same 2LD. In order to differentiate between a normal sub-domain and a sub-zone, the module traverses the labels from 2LD to the leftmost label. For each label, the NS resource record is requested to see whether that label is a sub-zone or not. If a sub-domain label has an NS record, it is a sub-zone under that 2LD. In the next step, the module profiles DNS zone activities of this sub-zone.

Profiling of DNS Zones: Understanding whether a sub-zone is used for payload distribution purposes can be achieved by analyzing its RR activities. These activities can be calculated as a function of access counts. By using the pDNS database, we extract all accessed RRs and their access counts. PDNS is built in a way that it counts the accesses to each RR for a certain period.

Let $R = R_A \cup R_{NS} \cup \cdots \cup R_{TXT}$ where $R_A = \{r_A \mid r_A \text{ is an A record}\}, \ldots, R_{NS} = \{r_{NS} \mid r_{NS} \text{ is a NS record}\}$ be the set of all RR types that can be defined in a DNS zone file, and $P = \{p \mid p \in (R \setminus R_{NS} \cup R_{CNAME})\}$ is the set of all the RR types that

are commonly used by payload distribution channels and other uses too. Since the TXT resource record is known to be the most suitable for payload distribution, we define a set $T = \{r_{TXT} \mid r_{TXT}$ is a TXT record$\}$ that holds any TXT record in a given zone.

For every RR type from sets P and T, the access count is retrieved from the pDNS database. Then, these access counts are aggregated to determine the μ value as formulated by Equation 3.2, which reflects the relation between the access ratios of T and P records as an indicator of payload distribution activities.

$$\mu = \frac{\sum_{i=0}^{n} ac_T}{\sum_{j=0}^{m} ac_P} \tag{3.2}$$

where n and m are the numbers of RRs for T and P respectively. Also ac_T and ac_P are access counts of each element in T and P respectively.

The Equation 3.2 provides a percentage value ($0 \leq \mu \leq 1$) of using TXT RR. When a domain name receives more access to RRs from P, it has a smaller μ value than a payload distribution channel domain that receives access only to TXT records.

Payload Distribution Intensity Analysis

Our payload distribution detection is based on DNS zone activities as described in Section 3.3.3. The Equation 3.2 provides the rating value of a domain name being used as payload distribution channel. However, it gives an imprecise and uncertain information about the intensity of the payload distribution channels. From an investigator perspective, detected domain names have to be prioritized based on their behaviors to facilitate the investigation process. When a domain name is abused,

it is important to learn its maliciousness intensity or severity level in a descriptive way. Fuzzy set theory can be used to transform the rating values to more descriptive meanings by introducing the level of intensity of the payload distribution of a given domain [91]. In Table 3.1, we have seven different levels of certainty, that are used in the estimative probability problem [54]. In general, when we have k levels, the parameters $\{p(L_1), p(L_1), \cdots, p(L_k)\}$ can be computed as a function of μ values in Equation 3.2 according to fuzzy triangular membership function [91]. Let each level L_t be a fuzzy subset, and each rating value μ is assigned to a membership grade $p(L_i, \mu)$ taking values in $[0, 1]$, with $p(L_i, \mu) = 0$ corresponding to non-membership in L_i, $0 < p(L_i, \mu) < 1$ to partial membership in L_i, and $p(L_i, \mu) = 1$ to full membership in L_i.

Level	Kent's Estimative terms	Probability
Very High (VH)	Certain	100%
High (H)	Almost Certain	93% ($\pm$ 6%)
Medium to High (MH)	Probable	75% ($\pm$ 12%)
Medium (M)	Chances About Even	50%($\pm$ 10%)
Low to Medium (LM)	Probably Not	30%($\pm$ 10%)
Low (L)	Almost Certainly Not	7%($\pm$ 5%)
Very Low (VL)	Impossible	0

Table 3.1: *Probability Value Scale [54]*

3.4 Dataset Collection

Throughout our experiments, we utilize three datasets to analyze the problem from different perspectives. Our datasets are a near-real time pDNS traffic, a pDNS database, and a malware database.

Passive DNS: In our experimental results, we evaluate the system on a one-month dataset, which spans between March 19, 2013 and April 19, 2013. According to the system logs, the total number of packets processed by our system is around 40 million packets with an average of about 1.3 million packets daily (Table 3.2).

Passive DNS Database: Our system also builds a pDNS database that stores all the data coming from the pDNS traffic. This database recorded the pDNS traffic that we utilize for profiling the DNS zone of domains.

Malware Database: We observe over one-year period of malware samples that are provided by a major security vendor. We receive the malware feed on a daily basis and then analyze each sample in a controlled environment to generate dynamic behavioral analysis reports. In our analysis, we just consider malware samples that conduct activities using the TXT RR for the DNS protocol. Table 3.2 shows some of the statistics about the malware feed recorded between January 2012 and December 2012.

3.5 Experimental Results

In this section, we explain the experimental results of our system. We show that our system can detect payload distribution channels in the pDNS traffic. The experimental results reveal long-running hidden domains, which are used by Morto worm to distribute attack payloads. We also found that on contrary to the common knowledge [65], some of these attack payloads are in clear text without any encoding and encryption. This indicates that our system can detect these channels regardless of the format of the distributed data.

Passive DNS	Period	30 days
	Number of DNS messages	20 Billion
	Number of TXT records	40 Million
Malware database	Period	1 Year
	Number of malware samples	15 Million
	Number of malware samples that have TXT activities	18 Thousand

Table 3.2: *Dataset Statistics*

During experiments, we tested our system with a live pDNS traffic for 30 days (Section 3.4). Domains that are accessed for TXT records are queued up in a time-based window; we set the window to be one day. When the window expires, the packets are fed to the query and response pattern module (Section 3.3.2). Once the patterns for each domain is recognized, these packets are sent for the DNS zone analysis (Section 3.3.3), to build the DNS zone profile for each zone. At the final step, the information gathered on each zone is passed to the intensity analysis module (Section 3.3.3), which uses a membership function to determine the level for each zone to be a payload distribution channel.

We used a computer with i7-2600 3.4 GHz CPU and 16 GB of RAM. The system performed well with near real-time analysis of the domains. The lookups to the

pDNS database did not create a significant overhead to the overall performance of the system.

3.5.1 Query and Response Patterns

In the first step of our system, we determine the query and response patterns on the captured traffic within one day. To evaluate the feasibility of each pattern to carry out payload distribution channels, Figure 3.3 compares the average distinct message counts from pDNS with the number of malware instances per pattern. Many-to-Many pattern can be considered as the best candidate for distributing large volume of data, while it was probed by small number of malware instances during the year of 2012. This extensive payload retrieval scheme would easily alert IDSs [77]. On the other hand, Single-to-Single pattern allows to carry small volume of data while maintaining a low network footprint. By observing our malware samples, we found that most of the malware instances used this pattern to retrieve the attack payloads. Because each of these instances send a single query to receive the attack payload, their queries can easily blend in the daily network traffic. Compared to other patterns, Single-to-Single is the best candidate to establish a fully resilient channel in DNS. Single-to-Many pattern requires to update the zone file to distribute different resource data for the same query. This is technically difficult to maintain because of the caching behavior of name servers. As we see in Figure 3.3, there is no single malware instances using this pattern. Although Many-to-Single pattern has a single response to the different queries like Single-to-Single pattern, it creates a large number of queries, that can be also noticed by IDSs.

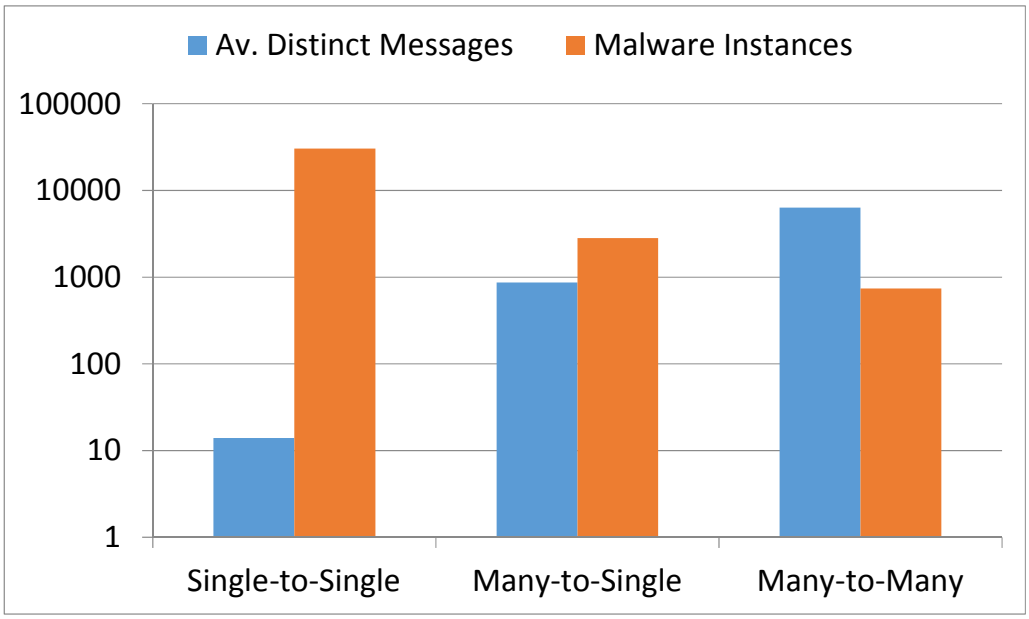


Figure 3.3: *Average Number of Query and Response Messages within Single Window (1 Day)*

As shown in Figure 3.3, Many-to-Many pattern is generating the most extensive traffic compared with the other patterns. The high volume of data exchange can reveal the name server, which is used as the payload provider. In order to hide this name server, botmasters are using it only for the bootstrapping phase to initiate payload distribution channels. The initial DNS query is directly made to a rogue name server without traversing the DNS hierarchy. Therefore, the malware authors could use any domain, even unresolvable domains. We found some instances of Feederbot botnet, which used known legitimate domains such as `yahoo.com`. However this first query is just the starting point of the payload distribution channel. In the first query, bot sends some data in sub-domain labels that are used as a key derivation parameter [29]. The response for this query comes as encoded with Base64, yet not encrypted. In this payload, the bot receives the domain name, that is considered to be the real

payload distribution channel domain, and an IP address of an open resolver to be used. The open resolver is abused for hiding the name server which provides the payload contents. Our observation confirms that open resolvers are often subject to different kinds of abuses [27]. After the bootstrapping process is completed, bot starts querying this domain to receive the attack payload in a sequential manner. Unlike what has been reported in [3], Feederbot only uses the unresolvable domains for bootstrapping process.

3.5.2 Payload Distribution Detection

When the query and response patterns of domain names are recognized, they are inspected by the DNS zone analysis module. The access counts of each RR of these domain names are gathered from the pDNS database. Equation 3.2 determines the μ values of each domain being used as payload distribution channel based on the access counts. During our experiments on the pDNS traffic, we have captured 2707 domains that have TXT resource record activities. Figure 3.4 shows the distribution of μ values across these domains.

To validate our system, we observe our malware dataset, and pDNS database to investigate the difference between payload distribution channels and regular domains. As regular domains, we use the top 500 domains from Alexa top sites[1], because they are used for different services. By using our 1-year malware dataset, we extracte malware domains, which are used for payload distribution. We retrieve the access counts for all RRs of each domain from regular and malware domains. These access counts are a good measure to understand the individual RR activity of any given

[1]http://www.alexa.com/topsites

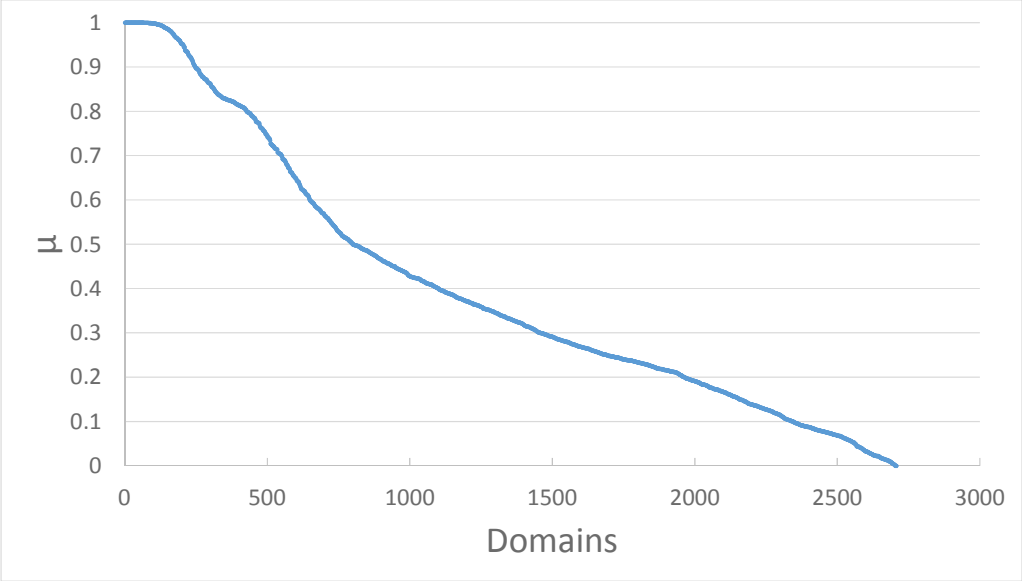


Figure 3.4: *Distribution of Rating Values of the Detected Domains*

domain. In Figure 3.5, the distribution of the access counts for these RRs is given. Domains from Alexa received DNS queries for different RRs. The reason for this could be the fact that these domains utilize DNS for enabling access to different services. On the contrary, malware domains received an extensive number of DNS queries for TXT records. These records are used to distribute the payload as it is the most suitable RR type within the protocol. We also investigate the access to the CNAME records in malware domains. They are used to redirect between malicious domains as botmasters maintain a network of malicious payload distribution channels.

When the system calculates μ values for all domains, it determines the intensity level of payload distribution of each domain. This step of the detection is based on the membership function, which was discussed in Section 3.3.3. The calculated μ values are fed to this intensity analysis mechanism where each domain is mapped to

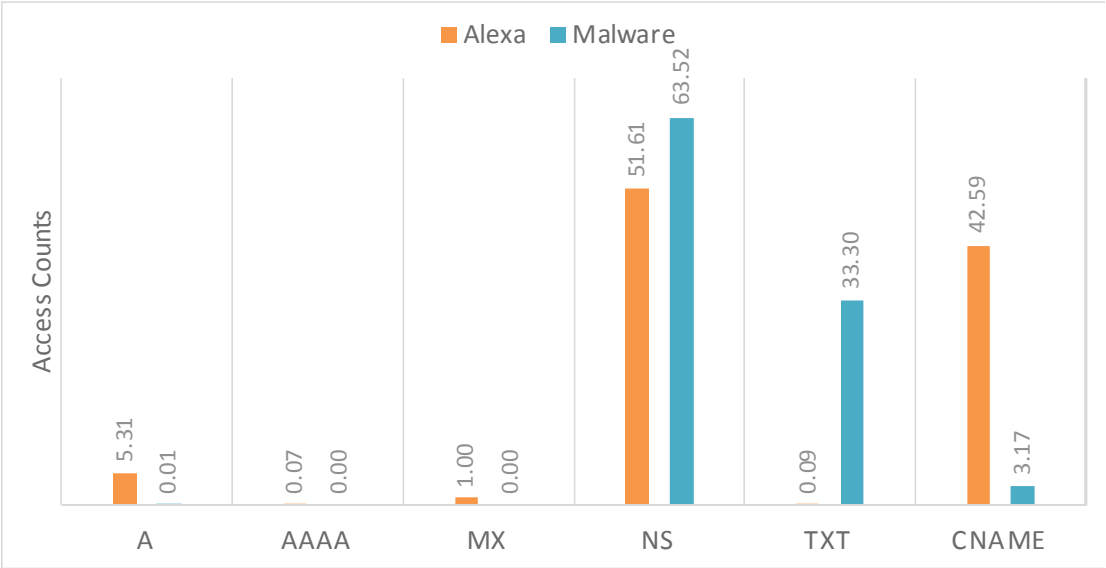


Figure 3.5: *Alexa and Malware Domains DNS Record Access Counts*

seven different levels. As a proof of concept, we determined seven levels in compliance with Kent's Estimative Terms [54].

Filteration Steps: There are some of the legitimate use cases that can behave as payload distribution channels. In fact, there are specifications that are using TXT records to apply some security measure for mail servers such as SPF, and DKIM (see Section 2.1.1). Since these specifications are designed for mail servers, the zone file should reflect the existence of MX RRs. As shown in Figure 3.5, malicious domains are not associated with any MX RRs. Therefore, these legitimate services can be recognized using two different filtration steps: MX RR activity, and specifications recognition.

The first filtration process takes each domain, and selects the most accessed TXT RR by using the pDNS database. Then, we apply a regular expression in the TXT record based on the defined syntax of specifications [5, 89] to determine any possible

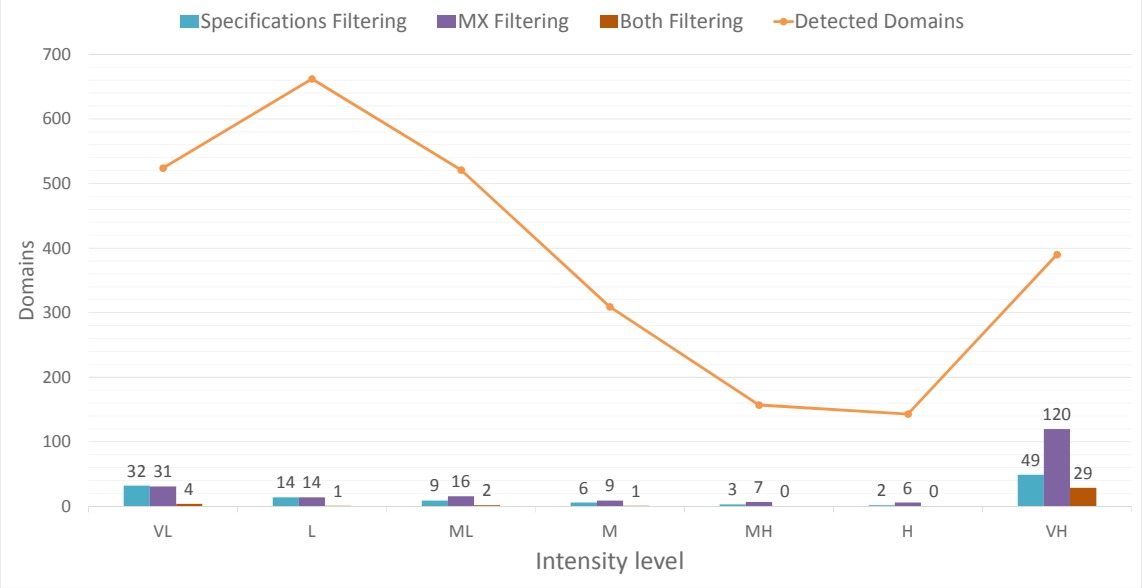


Figure 3.6: *Effect of Filtration Mechanisms on the Detected Domains*

specification string (e.g., SPF). In the second filtration process, we investigate the activities of MX RRs. When a domain name is associated with any MX RR activities, it is considered as non-payload distribution channel. In Figure 3.6, we show the distribution of domains across different levels of intensity as well as the results of filtration mechanisms. MX and specifications filtration mechanisms perform similar to each other in every intensity level. Compared to the others, level 7 has the most domains, which remained after applying both filtering mechanisms. Therefore, we focus on level 7, which is labeled as *Certain* as explained in Table 3.1. In Figure 3.7, the distinct domains that are detected as payload distribution channels are given in a daily basis across one-month along with the performance of each filtration step. The number of detected domains is 390 before any filtration is applied. However, some of these domains might be accessed mainly to receive specifications related data. The both filtration mechanism reduce the number of domains to one on average daily, and

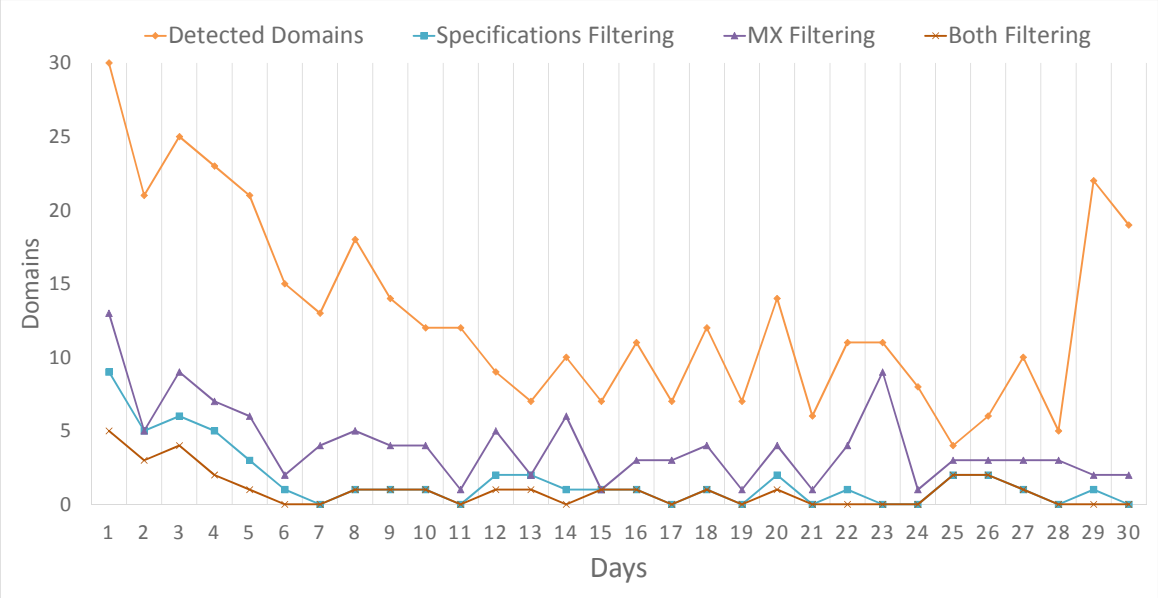


Figure 3.7: *A Daily Observation of the VH Intensity Level with the Filtration.*

we confirmed that the remaining domains are valid payload distribution channels as discussed ins Section 3.2.2.

The Resilient Morto Domains: Morto is a malware family that targets the Remote Desktop Protocol (RDP) to gain access to host machines. It is one of the malware families that use DNS as a payload distribution channel [65]. During our experiment, we detected domains that are being used by the Morto family. Morto uses the Single-to-Single pattern where a static query is sent to the malware domain to receive the encoded payload. In fact, it is known that it receives a Base64 encoded and encrypted URL, which points to the second payload [65]. We noticed that Morto domains also distribute IP addresses in clear text inside TXT records. A reverse lookup to one of these IP addresses in the pDNS database reveals that it is shared with other malicious domains. In Table 3.3, we give some statistics on the domains that are used by Morto instances. As mentioned in Section 3.5.2, the malware au-

thors also linked different domains to each other through CNAME records to maintain a malicious network. A simple investigation through the pDNS database reveals a network of domain names from Morto botnet.

Number of detected Morto domains	3
Life span of these domains	1.3 years on average
Average of access counts of TXT records	4080392

Table 3.3: *Statistics of Detected Morto Worm Domains*

3.6 Limitations and Discussions

To the best of our knowledge, our system has several limitations. The first limitation of our system is the inability to detect malware mimicking the DNS zone activities of legitimate domain names. We use the fact that name servers of payload distribution channels only receive requests for TXT records. If a domain is used for different malicious activities (e.g., spam, phishing) as well as for payload distribution, then it will be accessed for different RRs, e.g., A record for phishing scams. Then, our detection method might consider this domain as non-payload distribution domain.

The second limitation is that our system relies on our observations from our malware database. The malware database is established with dynamic analysis reports, which are generated in the sandbox for a limited time. It means that the dynamic analysis process might not capture all DNS communications of malware families. Therefore, we cannot know whether our system can detect all malicious payload distribution channels in DNS.

The third limitation is that our system is an offline detection mechanism which can detect after a domain is visible for an epoch. It cannot detect the payload

distribution channels as they become online. So, the system cannot be used for real time detection. However, it can detect them at the end of the epoch, which is one day in our experimental setup.

The last limitation of our system comes from pDNS. Its replication [87] is a unique way to collect the global DNS traffic by sensors. However, it has a shortcoming that might affect our results. Malware might not use caching resolver of the network and alternatively send the queries directly to an open resolver. In this case, the traffic would not pass through the sensor, and would not be analyzed. While this is a limitation of the pDNS replication mechanism, our system can detect payload distribution channels within the range of the existing pDNS stream.

Detection Regardless of Syntax: Our results show that the system detects payload distribution channels in DNS. As our method discover Morto domains: it also detects legitimate payload distribution channels as discussed in Section 3.2.2. It indicates that regardless of the syntax of the payload distribution channel, the DNS zone activity metric is a strong feature to detect domains, which are used for these channels. If botmasters start using a syntax similar to the legitimate services to blend in their traffic, they might not be detected by network monitors. However, our system still detects them because it monitors the DNS zone activities of payload distribution channels.

DNS Tunneling Detection: In the results of our experiments, we detected DNS tunneling activities from a single domain (a DNS tunneling app for Android). As our system is configured to monitor TXT records, it successfully detects any DNS tunneling activities on TXT records. If the tunnel is established by using another

RR type, we expect that our system would still detect as the detection is not based on the content of the RR, but the access counts of RRs.

Observations from Malware Database: In our malware dataset, we discover domain names that are used for payload distribution channels. The behavior of these malware samples introduced different methods to retrieve the malicious content as discussed in Section 3.2.2. One of the interesting ways is that they used indexed queries to receive attack payload in multiple response packets. Due to the size restriction on TXT RRs, the payload is chunked into parts and each part is placed in another TXT record. Bot clients start querying this series of packets in a sequential manner until the last packet is received. Some of these payloads are chunked up to thousands of packets. Surprisingly, this method is very similar to the patent from Trend Micro [57]. However, our results showed that this method is not seen in the pDNS anymore. There are two possible interpretations of not observing this behavior in our dataset. First, botmasters realized the significant exposure of using this behavior, which generates a large number of messages, then they decided to stop using it. Second, these domain names are directly resolved by their own name servers or other open resolvers, which are not captured by our pDNS sensors.

3.7 Conclusion

In this work, we shed some light on the abuse of the DNS protocol by malware for distributing attack payloads. We design a system that is able to characterize and detect the payload distribution channels within the pDNS traffic. Our system observes the DNS zone activities of a channel by gathering access counts of each RR

type, and determines the intensity of the payload distribution. By experimenting our system on the pDNS traffic during one month, we show that it detects the resilient malicious payload distribution channels, which were active more than 18 months. We find that most of the malware instances are using a resilient pattern to retrieve the attack payloads, because it can blend within the daily network traffic. Moreover, our system is able to detect payload distribution channels regardless of their syntax format.

Chapter 4

Passive DNS Database

This chapter describes our efforts on establishing a pDNS database. The design of the database is influenced by DNSDB of Farsight Security Inc. [34] DNS based analysis (for both research and investigation purposes) requires hands-on experience with real DNS data logs. This type of studies are often done by using local network DNS logs, however it is not enough to look at global trends in malicious activities. It is important to analyze global DNS activities to understand and design defense mechanisms against emerging threats, such as malicious payload distribution channels in DNS (see Chapter 3). To achieve that, we need to have a historical database of pDNS to correlate previous attacks with current ones to model the behaviors of malicious communication channels as well as other security threats. Therefore, first we will discuss the motivation behind this work. After that, we will describe the database with its technical details. Then, we will introduce the technologies used in this work, and the implementation details. Afterwards, the evaluation of the system will be given. Finally, we will provide concluding remarks.

4.1 Introduction and Motivations

DNS is an important part the global Internet traffic as discussed in Section 2.1.1. More than 250 million domains have been registered, and it is increasing every day [56].

The increasing use of DNS makes it a good vintage point for analyzing the global Internet use as well as emerging security threats.

Parallel to the increasing use of DNS, there is an increasing interest in DNS based threat analysis both in the academia and the industry. There is a significant shift to DNS for detecting global botnet trends [6, 7, 9, 12, 60]. Probably, the main reason for this is that botmasters have started to use domains rather than numerical addresses of their C&C servers. It is not surprising as domain names are more flexible from a botmaster's perspective. If a domain is taken down, it can be replaced with another domain, and in a few hours the C&C server would be back online. So, researchers propose techniques to observe botnets in different layers of DNS [7, 8]. This new trend in threat analysis makes it more difficult for botmasters to hide, and maintain their malicious networks. DNS based web application security is a new trend in the industry. It is a technique for defending websites against different types of threats, such as Structured Query Language (SQL) injection, DDoS, and web spammers. This defense mechanism becomes quite strong, even against targeted attacks. In the Spamhaus incident (see Section 1.1), Cloudflare, a web security company, deployed their DDoS protection systems at the DNS level to mitigate the biggest DDoS attack in the history [72]. Furthermore, Cloudflare offers other types of security measures in the DNS level.

It is clear that DNS is a good place to investigate and mitigate emerging threats, however it comes with some challenges. Compared to other types of threat analysis, DNS based threat analysis requires a different approach for logging the traffic. For example, in spam analysis, the spam data is simply captured by spamtraps [74]. It does not matter whether the spamtrap sensors are placed near the spammers's

geolocation. However, DNS data depends on where data capturing sensors are placed. If the sensors are placed in a network, then the captured DNS data is limited to that network, and it cannot be used for analyzing global trends in threats. So, sensors should be placed in multiple networks, especially in ISP networks. Considering the number of customers of an average ISP, capturing DNS traffic from ISPs definitely provides a large dataset. Another challenge of DNS is the growing data size. Godaddy, a well known web hosting company, deals with nearly 10 billion DNS queries per day [62]. If we consider Godaddy as an ISP; when the data is aggregated from multiple ISPs, it becomes challenging to process it.

Passive DNS comes into the play at this point. It is a strong DNS replication method, which aims to mirror DNS traffic for further analysis. Passive DNS inherits the challenges of capturing DNS traffic, therefore it has to address positioning of sensors, and an efficient solution for processing large captured datasets. It also needs to be near real-time for detecting threats as they show up in DNS. Deploying sensors in multiple networks, and collecting the captured data in a central data warehouse for further processing is a good approach. It is already applied by Farsight Security, Inc. [1] for SIE as discussed in Section 2.1.2. Their implementation includes a preprocessing pipeline, which sanitizes the captured data. This initiative has many sensors deployed in the US and Europe, and the collected DNS data is used in previous research works [7, 12].

The motivation behind our project initially comes from the need of a pDNS database in our work on detection of malicious payload channels in DNS. Also, there are other on-going research work in our Computer Security Laboratory that can benefit from a pDNS database. As mentioned in Section 1.3, after the completion of

writing this dissertation, another group from our lab has started working on the design and implementation of a pDNS database, which is based on a library (mtbl) [38] developed by Farsight Security, Inc.

Passive DNS comes as a stream of data. For analysis, it should be stored in a database in a way that it is efficient to query, and comprehensive to keep all data from the pDNS stream. The database also has to be scalable for distributing the load across multiple nodes without hassle. Initially, we deployed our database on a single node, which receives the pDNS feed. It can cope with the speed of the pDNS stream, while serving for queries without significant latency.

As a summary, the database provides following benefits:

1. **Historical database based on pDNS:** The database stores selected data from the pDNS stream in a NoSQL database system. It allows to retrieve any type of DNS related data.

2. **Scalable, efficient solution to store massive amount of DNS data:** It is based on a scalable database system, which makes the database easy to distribute across multiple nodes, e.g., a data center. Moreover, it is efficient enough to be used for live pDNS analysis (see Section 3.3.3).

3. **Easy to access database system:** We provide an API and a web interface access to the database. While the API provides a programmatic access to the database, the web interface gives the same functionality, which the API offers, with a user friendly interface.

4.2 Description

In this project, we address several requirements to accomplish for designing a fully fledged pDNS database system.

1. **Requirement 1: Easy access to the database:** Interaction with the database through programs, as well as a secure user interface.

2. **Requirement 2: Efficient database schema:** The database should store all of the pDNS data in an efficient and compact way. It important to keep what is needed to optimize the disk space usage.

3. **Requirement 4: Easy to scale:** Creating a cluster of database servers is often cumbersome, and it requires complex configurations. In our case, we cannot afford any halt in the system, because we have to process and log the data with zero loss. The database system should handle the clustering details without any degradation in write operations.

4. **Requirement 3: Easy to maintain and cost efficient:** Database systems often require advanced optimization skills. The database should require minimum effort for maintenance. Also, we should find an open-source database system for cost efficiency.

Initially, our first challenge is to analyze the pDNS stream channels, which are described in Section 2.1.2. As each channel has advantages and disadvantages, we need to reason our choice of channel. After choosing the right channel, we design our database based on the format of that stream. The database schema also relies

on the type of the database system that we have selected. In this project, we use a NoSQL database solution from Apache Software Foundation. We explain the reasoning behind using a NoSQL database as opposed to Relational Database Management Systems (RDBMSs). Finally we discuss the ways of querying the database.

Our project is to design and implement a pDNS database that can be built on top of a pDNS stream. In a pDNS system, it is expected to have a different snapshots of the processed DNS data, such as channels discussed in Section 2.1.2. It is important to choose the which state of the data to store in the database.

	Advantages	Disadvantages
The Input channel	• Raw data • Flexible	• Duplicated data • Cache poisoning data
The Output channel	• Sanitized data • Access counts for RRs	• Limited time-series analysis

Table 4.1: *Comparison between the Input and Output Channels*

Although we might have different channels, we have to choose between input and output channels as the remaining channels are only used to access to the internal stages of the pDNS processing pipeline (see Figure 2.4). The input channel is the entrance point of the pipeline, while output channel is the exit point. As seen in Table 4.1, both channels have advantages and disadvantages. The input channel is the first channel, and its output is raw data, which come from the capturing sensors. As it is raw, it comes with duplications, and potentially consists of cache poisoning data, which is the result of cache poisoning attacks on authoritative name servers. However, it gives full flexibility to apply different approaches for building our database. As

opposed to the processing pipeline from SIE, the final output could be more suitable for time series analysis. On the other hand, the output channel provides sanitized data, and as seen in Table 4.2, it provides extra information on top of existing DNS packet information[1]. Every RR name is processed in a window based system, in which every new RR name is pushed into the queue for a period. During this period, each time an existing RR name appears, its count is incremented. In this way, access count for each RR name is determined. However, this approach causes an inaccuracy for time series type of analysis. It is impossible to determine when each duplicating packet showed up in pDNS, because the timestamp information of each packet is overwritten by the next duplicating packet. Regardless of this fact, we choose the output channel for building our pDNS database, as the data is filtered from cache poisoning attacks as well as bailiwick is determined [30]. Considering the size of the pDNS data, implementing same functionalities goes beyond the scope of this project.

As the database system, we use Apache Cassandra project[2], which is a well-established open source project. It is a key-value based database system, which is very similar to flat file data structure. This new trend in database systems is called NoSQL, which is basically free from the complex structure of RDBMSs. The simplicity comes from the minimalistic organization of data. Each table corresponds to a query, which makes primary keys more important for queries. Therefore, we have to design our database schema according to the requirements of Cassandra.

As seen in Figures A.2 and A.3 in Appendix, our database schema is grouped under rrset and rdata. The design of these schemas are heavily influenced by the dnstable implementation from Farsight Security Inc. [37] Both table groups are also

[1]https://security.isc.org/NmsgType_SIE_dnsdedupe/
[2]http://cassandra.apache.org/

Field name	Purpose
type	the type of a packet based on its state in the pipeline
count	the access count for a packet. It is calculated during the packet's lifetime in the processing window
time_first	the time when a packet is first seen
time_last	the time when a packet is last seen
response_ip	the IP address of the authoritative name server, on which the sensor resides
rrname	the RR name of a packet
rrtype	the RR type of a packet
rrclass	the RR class of a packet
rrttl	the TTL value a packet
rdata	the response data of a packet
bailiwick	the DNS zone in which the rdata is given

Table 4.2: *Data Fields in the Output Channel*

grouped based on the format of their primary keys. This is the fundamental part of our database design. We consider two types of queries to our database: left-hand and right-hand wildcard queries as seen in Table 4.3. A left-hand side wildcard query seeks to find all RR names under a given domain, e.g., `*.google.com`, and a right-hand side wildcard query gets all RR names with a certain sub-domain, e.g., `news.*`. To support both query types, we have two types of primary keys in our tables. We take the RR name and reversed it for left-hand side queries. The original RR name

is used for right-hand side queries. Because Cassandra supports ordered keys, all primary keys in tables are ordered. Therefore, it becomes possible to get a slice from the table based on primary key range as seen in Table 4.3. It is important to mention that ranging is only possible if primary keys are in hexadecimal format. In this way, the query will be similar to getting numbers within a range, e.g., return numbers between 154 and 189. Primary keys are also appended with extra information from the DNS packet such as bailiwick, rrtype, and timestamp to make them unique to avoid collision with other primary keys.

Query type	Query result	Primary key range

Left-hand side
`*.google.com`
	sub-domains of the domain
	`news.google.com`
	`maps.google.com`
	⋮
		`\x03com\x06google\x00`
		`03636f6d06676f6f676c6500`
		⋮
		`\x03com\x06google\xff`
		`03636f6d06676f6f676c65ff`

Right-hand side
`news.*`
	domains with the sub-domain
	`news.google.com`
	`news.microsoft.com`
	⋮
		`\x04news\x00`
		`046e65777300`
		⋮
		`\x04news\xff`
		`046e657773ff`

Table 4.3: *Wildcard Queries (primary keys are in original padded format of RR names)*

One of our requirements is to make the database accessible by different methods. We provide two access methods: an API and a web interface as seen in Figure 4.1. The first one is for querying the database directly from programs, the latter one is to give a quick and visual access to database. The API should be easy to access, so

we use Hypertext Transfer Protocol Secure (HTTPS) requests. The parameters of the request provide the options for the query, and an API key authorizes the request. The web interface is useful when investigating an incident, its user-friendly and secure (HTTPS) interface provides a fast access to the database.

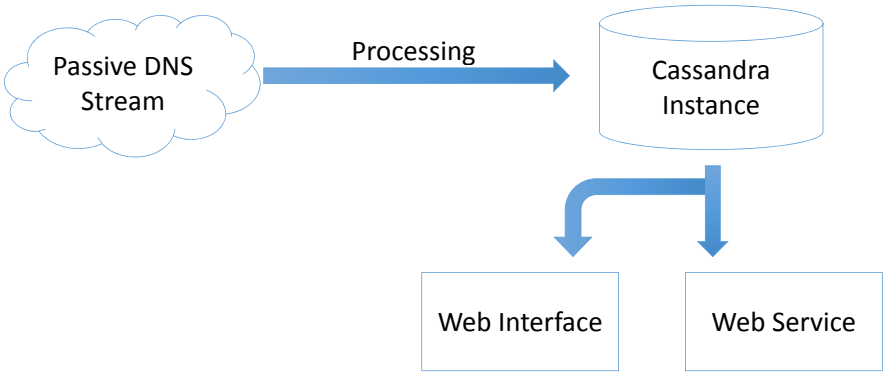


Figure 4.1: *Passive DNS Database Overview*

4.3 Preliminaries

The pDNS database is a combination of existing open-source tools and libraries. As a requirement of the project, we maintain the cost-efficiency without compromising any functionality. In this section, we discuss the tools that we utilize to build the database. The implementation is done by using Python. It is used in for both building and querying the database. One of the main reasons to choose Python is the availability of the libraries for this language. Finally, we use Apache Cassandra as our database system, which is designed for fast read and writes.

We use different libraries for reading and writing. As seen in Figure 4.2, there is a dependency between the libraries. We use the packet format nmsg [39], which is designed and developed by Farsight Security, Inc. [1] It is built on top of the pcap

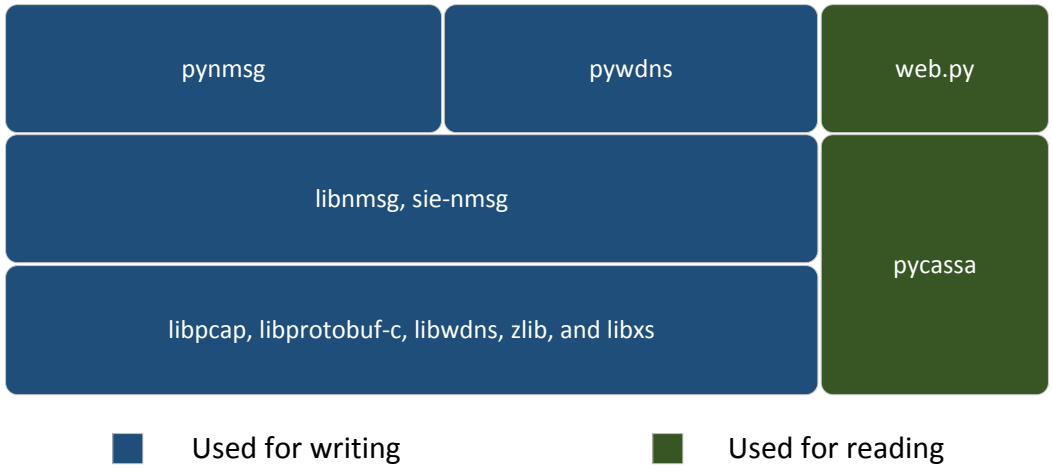


Figure 4.2: *Libraries Used in this Project*

format, so libpcap[3] is required for parsing this packet format. As an improvement over the time, Farsight Security, Inc. [1] adapted Google's libprotobuf[4] library for transferring the data over the wire. Protocol buffers (libprotobuf) is a powerful library, which is designed for Google's internal communication systems. Its data structure allows transferring data in a more compact way, which results performance improvements compared to other libraries. Nmsg specifically relies on the C extension of the library, which is libprotobuf-c. For de-serializing DNS packets, libwdns is developed by Farsight Security, Inc. [42] Additionally, libnmsg depends on zlib[5] and libxs[6] for decompressing and messaging purposes respectively. On top of libnmsg, sie-nmsg library is required to parse dnsdedupe packet format. This is a specific format for some channels in pDNS. Dnsdedupe format consists of the fields given in Table 4.2 along with some extra operational fields. Since the implementation is in Python, we use pywdns [41] and pynmsg [40], which are Python extensions of libwdns

[3]http://www.tcpdump.org/
[4]https://developers.google.com/protocol-buffers/
[5]http://www.zlib.net/
[6]https://github.com/crossroads-io/libxs

and libnmsg respectively. To interact with our Apache Cassandra instance, we use a client library pycassa[7], which is written for Python. Finally we use web.py[8] for developing the API as well as the web interface of our pDNS database system.

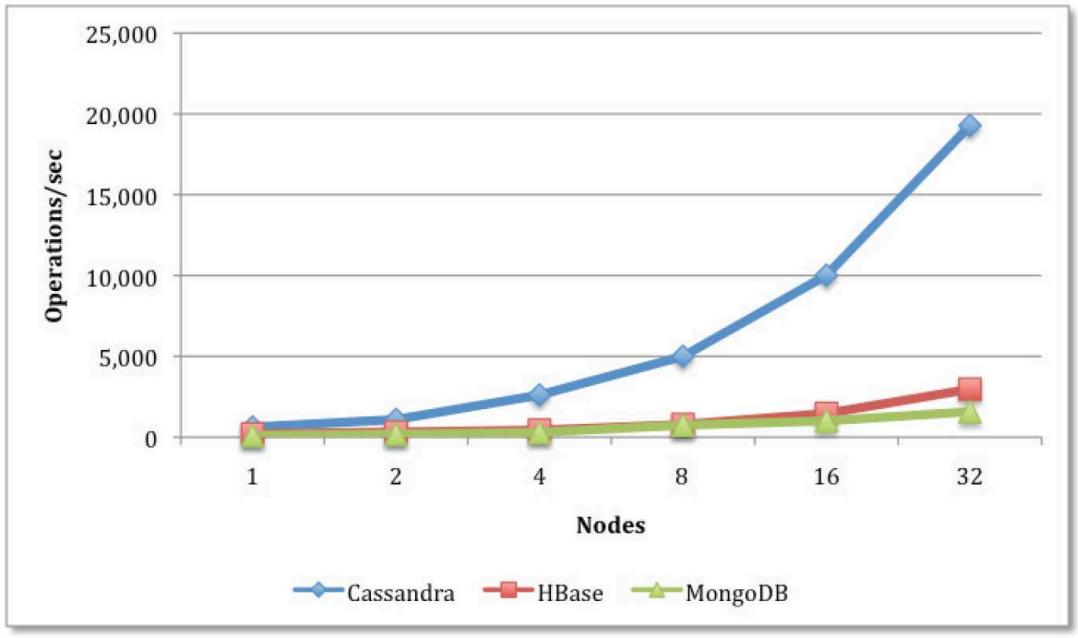


Figure 4.3: *Cassandra read workload comparison with HBase and MongoDB [51]*

As mentioned earlier, we decide to use an open-source NoSQL database solution for this project. NoSQL is a new phenomenon among database systems. It promotes the use of key-value based data structures for faster writing and reading. It is similar using a flat file based storage mechanism, which is basically writing the data in files. As opposed to writing sequentially to the disk, NoSQL databases introduce intelligent algorithms for handling the data in the memory prior to flushing it to the disk. This type of memory-based operations can also be done using SQL database systems. However, NoSQL database systems differ from the architectural perspective.

[7]https://github.com/pycassa/pycassa
[8]http://webpy.org/

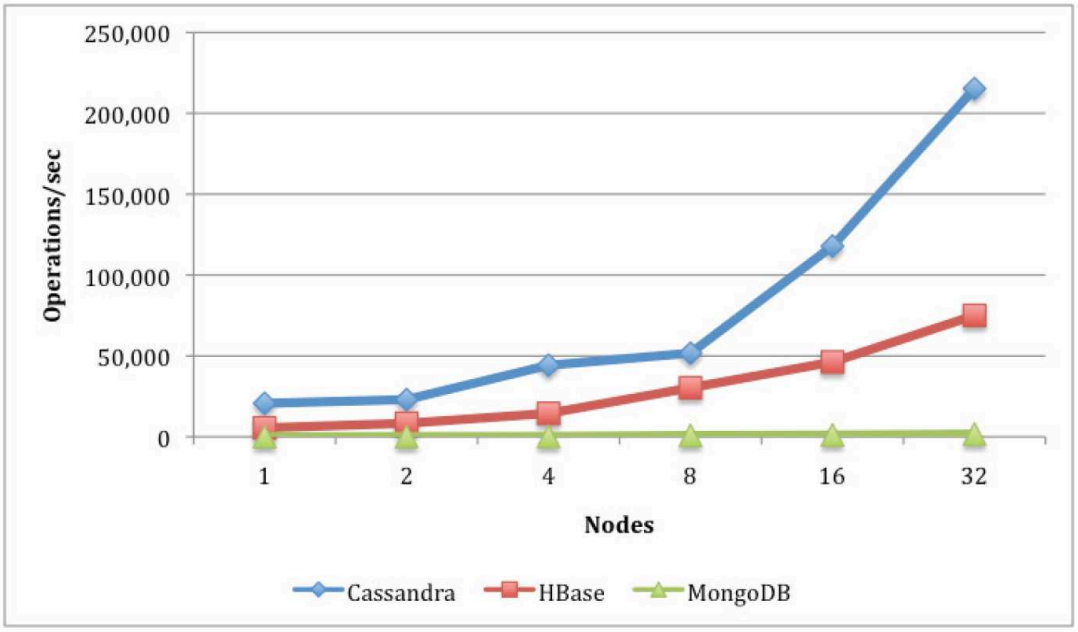


Figure 4.4: *Cassandra write workload comparison with HBase and MongoDB [51]*

The tables are organized according to the query, as opposed to relational databases. Therefore, there is no need for joining tables, hence faster queries. They are a better solution for data logging mechanisms, rather than complex web applications. For example, a social network website would require a complex relation between tables, and achieving this with a NoSQL table might not be feasible. For our project, Cassandra was the best option compared to other NoSQL solutions [51]. From the scalability point of view, it scales well across multiple nodes with an increasing performance for both reading and writing operations proportional to the size of the cluster (see Figure 4.3, 4.4).

4.4 Implementation

The implementation is considered in two folds: storing the pDNS data and querying it from the database. The process of storing the data comes with some performance challenges. Our pDNS streams constitute tremendous amount of data per second, and writing it to the database requires well optimized processing steps. Reading data from the database is reasonably less challenging compared to writing to the database. It is based on a central querying logic, which is serving to the API and the web interface.

4.4.1 Storing the data

Our mission is to implement a mechanism, which reads from a constant stream of data, processes it, and finally stores it into a database. We chunk our implementation into these three steps with the same order. First, we have to implement the logic that will be hooked on the pDNS VLAN for reading the packets, as they come from the socket. Each nmsg packet has to be parsed with libnmsg to extract the desired fields. After this step, we need to process these fields to fit in our data structure. It requires using libwdns for tackling DNS data. Finally we establish connections to our database server for writing data.

In Figure 4.5, the flow of the storing process is given. We use IO object of pynmsg for creating a reader on the channel. This reader maintains a constant read on the channel, and passes the input to a callback function for further processing. This part of the implementation is straightforward. In the callback function, packets are pushed into a First In First Out (FIFO) queue. We only take nmsg packets with the type

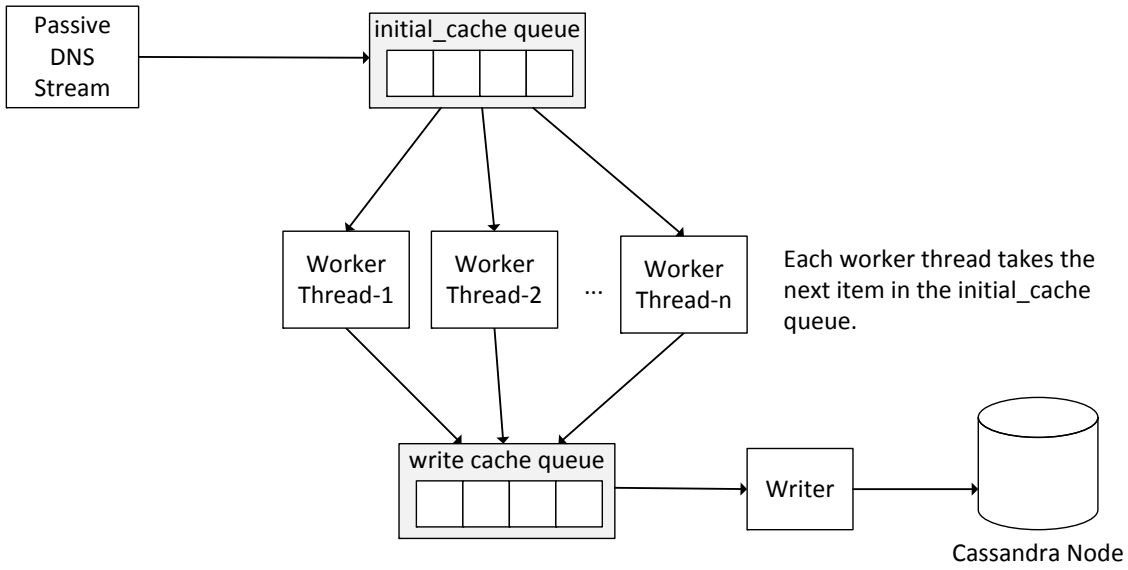


Figure 4.5: *Flow of the Writing Process*

"EXPIRATION". This information is used when DNS packets are processed through SIE pDNS channels. When a packet shows up in the processing pipeline, it is pushed into a queue as "INSERTION", which means that the packet has been seen for the first time. Finally, when the processing window for that packet expires, it is marked as "EXPIRATION" [30]. Therefore, this packet is the final version of that particular query, and we store this final version in our database. In our queue, these packets are aligned for the multi-threaded preprocessing.

During the preprocessing of packets, we extract fields from them, and prepare the field data for writing to the database. The main load of our program comes from this step, therefore we distribute the tasks among multiple threads. At this step, primary keys are composed, also RR name, RR type, RR class, and RR bailiwick data are converted from padded hexadecimal format to string by using pywdns. RR data packets are also prepared with corresponding primary keys. Also, reversing for primary keys is only applied to RR data with RR type NS and CNAME. These RR

types carry their data only in FQDN format, and it enables running wildcard queries for them. However, other RR types are not suitable for reversing their RR data. For example, we cannot expect a wildcard query on IP addresses, which are stored in A or AAAA RR types.

Parameter	Value	Purpose
initial_cache_size	4000	the size of the first queue, which stores nmsg packets
write_cache_size	4000	the size of the second queue, which stores preprocessed packets
flush_size	50	the number of packets to be stored in memory prior to flushing to the disk
worker_count	5	the number of threads for preprocessing
write_sleep_time	15	the number of seconds allowed Cassandra for garbage collection
log_freq_seconds	30	logging frequency

Table 4.4: *Configuration Parameters of the Writing Process*

As seen in Figure 4.4, we use a single thread for writing to the database. The reason of this approach is that we initially use a single node as our database server, and we need to avoid potential locks caused by multiple write operations. After adding more nodes to the cluster, we can have extra threads to increase the write throughput.

4.4.2 Querying the database

Our pDNS database provides two interfaces for querying as seen in Figure 4.1. The API is aimed to serve to queries directly from programs, especially for automatic accesses. To query the database, a URL must be composed of the IP address of the web server, which runs the API and the web interface scripts, along with the query parameters. A request to this URL should be sent through HTTPS for a secure connection to mitigate eavesdropping. Our goal is to maintain a similar functionality between the API and the web interface, therefore we use the same set of parameters for the web interface. These two interfaces return the same set of data for the same parameters without any modification in the result set.

Table 4.5 describes query parameters. These parameters are heavily influenced from the parameters of DNSDB of Farsight Security Inc. [35] By using these parameters in queries, the database can be queried with simple HTTPS requests. The result is returned in JavaScript Object Notation (JSON) format. This object notation format is similar to XML, which allows serializing the data for passing along the wire. It also simplifies parsing the data with native Python libraries. As seen in Figure A.4, a simple Python code snippet can be used to query the database. The ouput is similar to the output of the pDNS database solution (DNSDB) developed by Farsight Security, Inc. [35, 36]

The web interface is designed to be a simple search page with security measures. It is designed with HTML and CSS to have a user-friendly interface. Initially a welcome page welcomes the user for logging into the system. After verifying the user credentials, the search page is presented. It provides the same parameters, as the

Parameter	Value	Purpose
search_type	rrset or rdata	Determines the type of the search whether it is for RR name, or RR data.
input_mode	Name, IP, raw	Determines the type of the keyword.
Keyword	String value	Depending on keyword_type, it can be a wildcard query, a network IP space with slash notation (e.g., 1.1.1.1/16), or raw hexadecimal data.
rrtype	A, AAA, etc.	Filters the query based on RR type of DNS packets.
bailiwick	FQDN	Filters the query based on bailiwick data.
limit	Positive integer value	Limits the number of results to be returned, 0 returns all results.

Table 4.5: *Query Parameters*

API, in a search box, and the results will be presented in a table below the search box.

4.5 Evaluation

The evaluation of our project is based on our requirements, which are the performance of storing and querying the pDNS data. In the ideal scenario, we would expect a similar benchmark results to Cassandra benchmark results. However, it is expected that the processing of the pDNS data will add an extra overhead. The retrieval of the data also requires some additional computation, which is also expected to affect the

benchmark results. The additional computation occurs due to the filtration of the retrieved data. Because the database only accepts queries on primary keys, additional filtering is applied on the retrieved data.

Node Specifications	**CPU** i7-2600 3.4 GHz **RAM** 16 GB **OS** Ubuntu 12.04 LTS **Java** Oracle Java 1.7.0_45
Number of nodes	1
Project Configuration	Default (see Table 4.4)
Cassandra Version	1.1.6
Cassandra partitioner	org.apache.cassandra.dht.ByteOrderedPartitioner
Number of records	1M

Table 4.6: *Evaluation Setup*

The results of the evaluation is presented in Table 4.7. The official Cassandra results [51] are tested on a similar node setup as seen in Table 4.6, therefore the comparison with these results can reflect a true benchmark for the performance of our system. As mentioned earlier, the comparison is aimed to indicate the overall overhead of our system when the Cassandra results are subtracted. The read throughput of our system is an expected result, because the data goes under extra processing after it is retrieved from the database. The load of this process varies depending on the

Test	Our system (rows/sec)	Apache Cassandra (rows/sec)	Loss by Process overhead (rows/sec)
Read	552.49 ± 23	624.31	71.82 ± 23
Write	19028.41	20692.26	1,663.85
Read and Update	409.21 ± 11	467.34	58.13 ± 11

Table 4.7: *Throughput of our Project with the Official Benchmark Results of Cassandra*

query, therefore a throughput range occurred between different query types. The data is filtered according to the rrtype, and this filtration occurs on the data, which is returned from the database. The write throughput is constant as every packet from the pDNS channel is processed in the same way before it is written to the database. The write producess overhead due to the preparation of the pDNS data to write into the database. Finally, we also test the system concurrent read and update operations. For this part of the evaluation, we update existing nodes while we query the database. The system shows 58.13 ± 11 performance overhead compared to Cassandra, which is the result of extra processing.

The overhead, which is caused by the processing of data, is an inevitable result of the project. As this is expected, we do not consider the processing overhead as a limitation of the system. We rather take these measurements as indicators to optimize the processes to decrease the overhead. Also, Cassandra can compensate the overhead by adding more nodes to the system (see Figure 4.3 and Figure 4.4).

4.6 Conclusion

In this chapter, we present the design and implementation of our pDNS database. As DNS becomes a part of threat analysis more and more, pDNS is the ultimate solution for replicating the global DNS traffic. We design a system that can match with needs of an efficient, scalable pDNS database system, which is completely built with open-source tools and systems. It is built on top of a pDNS data stream, and stores all DNS data without any modification. It uses a key-value based NoSQL database system, which is proven to handle constant read and writes, while scaling across multiple nodes, even multiple clusters.

Although the evaluation in the testing environment shows promising results, we have found that Cassandra has serious shortcomings. We experienced degradation in the write process due to the internal housekeeping processes by Cassandra. As a result of this problem, my colleagues in NCFTA has started implementing the database using mtbl system [38] developed by Farsight Security, Inc.

Chapter 5

Conclusion and Future Work

Botnets keep evolving, and botmasters come up with new solutions to mitigate existing defense technologies. This endless game results novel security threats, and they are often based on vulnerabilities in software systems. In this dissertation, we have discussed a security vulnerability, which exists in DNS. It enables data transfer by using DNS query and response packets, which go undetected, because the DNS traffic is considered to be trustworthy by network administrators. The DNS traffic is often unmonitored, and botmasters use this advantage of the protocol to send and receive their attack payloads. We have found different types of malicious communication channels in DNS, and introduce a novel technique to detect them regardless of the encryption state of the traffic. During our experiments, we found that there are several long-running domains, which are used for malicious payload distribution networks. We also found that some payloads are sent in clear text as opposed to what is reported by previous works. As long as this vulnerability within the DNS protocol is not fixed, malware families will keep exploiting it increasingly to establish resilient malicious networks.

The correlation between existing and emerging threats require a comprehensive analysis of historical datasets. In DNS based research, the importance of pDNS is obvious. However, pDNS comes with its own challenges in terms of the amount of the data. Especially, when it comes to storing the data in a database for further analysis.

In this dissertation, we propose a pDNS database architecture, and implement it to be used for different types of DNS based research works. Our database system keeps up with the speed of the pDNS stream without any significant degradation.

During our investigation on the malicious payload distribution channels in DNS, we have discovered some research directions that can be explored as an extension to the work presented in this dissertation.

1. **Detection of multi-purpose malicious domains:** Our proposed solution might not be able to detect domains, which are used for different malicious purposes. Our DNS zone based metrics can be extended with a semantic-aware approach to investigate the traffic content by using some metrics proposed in DNS tunnel detection studies.

2. **Detection of rogue DNS resolvers:** During our analysis, we noticed some rogue open resolvers that are used for querying malicious domains. We believe that these rogue resolvers are specifically chosen by botmasters, because they are operated with loose security policies. These resolvers can be included as a metric into our system for more accurate decision making process.

3. **Flexible pDNS database architecture:** Our pDNS database is scalable, however it can be extended with another layer of database, which would allow more flexible queries. Because we use a key-value based database solution, the schema of the database is forced to be very simple. Therefore, queries are more predetermined compared to queries in relational database systems.

Bibliography

[1] Security Information Exchange (SIE), Farsight Security Inc. https://www.farsightsecurity.com.

[2] The role of DNS in botnet command & control. Technical report, OpenDNS, 2012. http://info.opendns.com/rs/opendns/images/OpenDNS_SecurityWhitepaper-DNSRoleInBotnets.pdf.

[3] DNS for massive-scale command and control. *IEEE Transactions on Dependable and Secure Computing*, 10(3):143–153, 2013.

[4] Maurizio Aiello, Alessio Merlo, and Gianluca Papaleo. Performance assessment and analysis of DNS tunneling tool. *Logic Journal of IGPL*, 21(4):592–602, 2013.

[5] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. Domainkeys identified mail (DKIM) signatures. RFC 4871, May 2007.

[6] Manos Antonakakis, David Dagon, Xiapu Luo, Roberto Perdisci, Wenke Lee, and Justin Bellmor. A centralized monitoring infrastructure for improving DNS security. In *Recent Advances in Intrusion Detection (RAID '10)*, volume 6307, pages 18–37. Ottawa, Ontario, Canada, September 2010.

[7] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for DNS. In *USENIX Security Symposium*, Washington, DC, USA, August 2010.

[8] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou II, and David Dagon. Detecting malware domains at the upper DNS hierarchy. In *USENIX Security Symposium*, San Francisco, CA, USA, August 2011.

[9] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: detecting the rise of DGA-based malware. In *USENIX Security Symposium*, Bellevue, WA, USA, August 2012.

[10] Broderick Aquilino and et al. F-secure threat report H2. Technical report, F-Secure, 2012. http://www.f-secure.com/static/doc/labs_global/Research/Threat_Report_H2_2012.pdf.

[11] Dušan Bernát. Domain name system as a memory and communication medium. In *SOFSEM 2008: Theory and Practice of Computer Science*, volume 4910, pages 560–571. Springer, 2008.

[12] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Network and Distributed System Security Symposium (NDSS '11)*, San Diego, California, USA, February 2011.

[13] Hamad Binsalleeh, Thomas Ormerod, Amine Boukhtouta, Prosenjit Sinha, Amr Youssef, Mourad Debbabi, and Lingyu Wang. On the analysis of the Zeus botnet crimeware toolkit. In *Conference on Privacy Security and Trust (PST '10)*, Ottawa, Ontario, Canada, August 2010.

[14] Kenton Born. Browser-based covert data exfiltration. In *Annual Security Conference*, Las Vegas, NV, USA, April 2010.

[15] Kenton Born. Browser-based covert data exfiltration. In *Black Hat*, Las Vegas, NV, USA, July 2010.

[16] Kenton Born and David Gustafson. Detecting DNS tunnels using character frequency analysis. In *Annual Security Conference*, Las Vegas, NV, USA, April 2010.

[17] Kenton Born and David Gustafson. Ngviz: detecting DNS tunnels through n-gram visualization and quantitative analysis. In *Annual Workshop on Cyber Security and Information Intelligence Research*, Oak Ridge, Tennessee, USA, April 2010.

[18] Seth Bromberger. DNS as a covert channel within protected networks. Technical report, National Electronic Sector Cyber Security Organization, 2011. http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/DNS_Exfiltration_2011-01-01_v1.1.pdf.

[19] Patrick Butler, Kui Xu, and Danfeng Daphne Yao. Quantitatively analyzing stealthy communication channels. In *Conference on Applied Cryptography and Network Security (ACNS '11)*, Nerja, Spain, June 2011.

[20] Alper Caglayan, Mike Toothaker, Dan Drapeau, Dustin Burke, and Gerry Eaton. Real-time detection of fast flux service networks. In *Cybersecurity Applications and Technology Conference for Homeland Security (CATCH '09)*, Washington, DC, USA, March 2009.

[21] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet detection by monitoring group activities in DNS traffic. In *Conference on Computer and Information Technology (ICCIT '07)*, Aizu-Wakamatsu, Fukushima, Japan, October 2007.

[22] Lucian Constantin. Malware increasingly uses DNS as command and control channel to avoid detection, experts say. Published on February 29, 2012. Retrieved on August 1, 2013. http://www.networkworld.com/news/2012/022912-malware-increasingly-uses-dns-as-256763.html.

[23] Luciana Costa and Roberta DAmico. Malware detection and prevention platform: Telecom italia case study. In *ISSE 2010 Securing Electronic Business Processes*, pages 203–213. Berlin, Germany, October 2011.

[24] David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P. Lee, and Wenke Lee. Recursive DNS architectures and vulnerability implications. In *Network and Distributed System Security Symposium (NDSS '09)*, San Diego, CA, USA, February 2009.

[25] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased DNS forgery resistance through 0x20-bit encoding: security via leet queries. In *ACM Computer and Communications Security (CCS'08)*, Alexandria, VA, USA, October 2008.

[26] David Dagon and Wenke Lee. Global internet monitoring using passive dns. In *Cybersecurity Applications and Technology Conference for Homeland Security (CATCH '09)*, Washington, DC, USA, March 2009.

[27] David Dagon, Niels Provos, Christopher P Lee, and Wenke Lee. Corrupted DNS resolution paths: The rise of a malicious resolution authority. In *Network and Distributed System Security Symposium (NDSS '08)*, San Diego, California, USA, February 2008.

[28] Luca Deri, Lorenzo Luconi Trombacchi, Maurizio Martinelli, and Daniele Vannozzi. Towards a passive dns monitoring system. In *Symposium on Applied Computing (SAC '12)*, Riva del Garda, Italy, December 2012.

[29] Christian J. Dietrich, Christian Rossow, Felix C. Freiling, Herbert Bos, Maarten van Steen, and Norbert Pohlmann. On botnets that use DNS for command and control. In *European Conference on Computer Network Defense (EC2ND '11)*, pages 9–16, Gothenburg, Germany, September 2011.

[30] Robert Edmonds. SIE passive DNS architecture. Technical report, Farsight Security Inc., 2012. https://archive.farsightsecurity.com/Passive_DNS/passive-dns-architecture.pdf.

[31] Wendy Ellens, Piotr uraniewski, Anna Sperotto, Harm Schotanus, Michel Mandjes, and Erik Meeuwissen. Flow-based detection of dns tunnels. In *Emerging Management Mechanisms for the Future Internet*, volume 7943, pages 124–135. Barcelona, Spain, June 2013.

[32] Nicolas Falliere and Eric Chien. Zeus: King of the bots. Technical report, Symantec, 2009. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/zeus_king_of_bots.pdf.

[33] Greg Farnham. Detecting DNS tunneling, February 2013. http://www.sans.org/reading_room/whitepapers/dns/detecting-dns-tunneling_34152.

[34] Farsight Security Inc. DNSDB. https://www.dnsdb.info.

[35] Farsight Security Inc. DNSDB API. https://api.dnsdb.info/.

[36] Farsight Security Inc. DNSDB API access sript. https://raw.github.com/dnsdb/dnsdb-query/master/dnsdb_query.py.

[37] Farsight Security Inc. Dnstable. https://github.com/farsightsec/dnstable/blob/master/man/dnstable-encoding.5.txt.

[38] Farsight Security Inc. Immutable sorted string table library (mtbl). https://github.com/farsightsec/mtbl.

[39] Farsight Security Inc. Nmsg library. https://github.com/farsightsec/nmsg.

[40] Farsight Security Inc. Pynmsg library. https://github.com/farsightsec/pynmsg.

[41] Farsight Security Inc. Pywdns library. https://github.com/farsightsec/pywdns.

[42] Farsight Security Inc. Wdns library. https://github.com/farsightsec/wdns.

[43] Sean Gallagher. How the most massive botnet scam ever made millions for estonian hackers. Published on November 10, 2011. Retrieved on August 1, 2013. http://arstechnica.com/tech-policy/2011/11/how-the-most-massive-botnet-scam-ever-made-millions-for-estonian-hackers/.

[44] Francisco J. Gomez and Carlos Juan Diaz. Preventing use nameservers as malware distribution platform. Work in progress, January 2012. http://tools.ietf.org/html/draft-cmd-prevent-malware-dns-distribute-00.html.

[45] John Gordon. Systems and methods for identifying a network. Patent No. US8353007 B2, Filed Oct. 13th., 2009, Issued Jan. 8th., 2013, 2009.

[46] Shuang Hao, Nick Feamster, and Ramakant Pandrangi. Monitoring the initial DNS behavior of malicious domains. In *Conference on Internet measurement conference (IMC '11)*, Berlin, Germany, November 2011.

[47] Wesley Hardaker. Requirements for management of name servers for the DNS. RFC 6168, May 2011.

[48] Thorsten Holz. A short visit to the bot zoo [malicious bots software]. *IEEE Security & Privacy*, 3(3):76–79, 2005.

[49] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. Measuring and detecting fast-flux service networks. In *Network and Distributed System Security Symposium (NDSS '08)*, San Diego, CA, USA, February 2008.

[50] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET '08)*, San Francisco, CA, USA, April 2008.

[51] Datastax Inc. Benchmarking top NoSQL databases, 2013. http://www.datastax.com/resources/whitepapers/benchmarking-top-nosql-databases.

[52] Jian Jiang, Jinjin Liang, Kang Li, Jun Li, Haixin Duan, and Jianping Wu. Ghost domain names: Revoked yet still resolvable. In *Network and Distributed System Security Symposium (NDSS '12)*, San Diego, CA, USA, February 2012.

[53] Dan Kaminsky. Black Ops of DNS. In *Black Hat Briefings*, Las Vegas, NV, USA, July 2004.

[54] Sherman Kent. Words of estimative probability. *Studies in Intelligence*, 8(4):49–65, 1964.

[55] Ed M. Kucherawy. Domain-based message authentication, reporting and conformance (DMARC), March 2013.

[56] Frederic Lardinois. Report: More than 250m domain names have now been registered, almost half are .com and .net. Published on April 8, 2013. Retrieved on August 1, 2013. http://techcrunch.com/2013/04/08/internet-passes-250m-registered-top-level-domain-names/.

[57] Jianda Li, Bharath Kumar Chandrasekhar, and Kong Yew Chan. Updating of malicious code patterns using public DNS servers. Patent No. US8171467 B1, Filed Jul., 3th., 2007, Issued May, 1st., 2012, 2007.

[58] Samuel Marchai, Jérôme François, and Thomas Engel. Semantic based dns forensics. In *Workshop on Information Forensics and Security (WIFS '12)*, Tenerife, Spain, December 2012.

[59] S. Marchal, J. Francois, C. Wagner, R. State, A. Dulaunoy, T. Engel, and O. Festor. DNSSM: A large scale passive DNS security monitoring framework. In *Network Operations and Management Symposium (NOMS '12)*, Maui, HI, USA, April 2012.

[60] Samuel Marchal and Thomas Engel. Large scale DNS analysis. In *Dependable Networks and Services*, volume 7279, pages 151–154. Luxembourg, Luxembourg, June 2012.

[61] Alessio Merlo, Gianluca Papaleo, Stefano Veneziano, and Maurizio Aiello. A comparative performance evaluation of DNS tunneling tools. In *Conference on Computational Intelligence in Security for Information Systems (CISIS '11)*, pages 84–91, Torremolinos-Málaga, Spain, June 2011.

[62] Rich Miller. How go daddy keeps 52 million domains running. Published on April 17, 2012. Retrieved on August 1, 2013. http://www.datacenterknowledge.com/archives/2012/04/17/how-go-daddy-keeps-52-million-domains-humming/.

[63] Paul Mockapetris. Domain names: concepts and facilities. RFC 1034, November 1987.

[64] Paul Mockapetris. Domain names: implementation and specification. RFC 1035, November 1987.

[65] Cathal Mullaney. Morto worm sets a (DNS) record. Technical report, Symantec, 2011. http://www.symantec.com/connect/blogs/morto-worm-sets-dns-record.

[66] Jose Nazario and Thorsten Holz. As the net churns: Fast-flux botnet observations. In *Conference on Malicious and Unwanted Software (MALWARE '08)*, Fairfax, VA, USA, October 2008.

[67] Lucas Nussbaum, Pierre Neyron, and Olivier Richard. On robust covert channels inside DNS. In *Information Security Conference (SEC '09)*, volume 297, pages 51–62. Pafos, Cyprus, May 2009.

[68] Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. FluXOR: detecting and monitoring fast-flux service networks. In *Detection of intrusions and malware, and vulnerability assessment (DIMVA '08)*, volume 5137, pages 186–206. Paris, France, July 2008.

[69] Vern Paxson, Mihai Christodorescu, Mobin Javed Josyula Rao, Reiner Sailer, Douglas Schales, Marc Ph Stoecklin, Kurt Thomas Wietse Venema, and Nicholas Weaver. Practical comprehensive bounds on surreptitious communication over DNS. In *USENIX Security Symposium*, Washington, DC, USA, August 2013.

[70] Roberto Perdisci, Igino Corona, David Dago, and Wenke Lee. Detecting malicious flux service networks through passive analysis of recursive DNS traces. In *Annual Computer Security Applications Conference (ACSAC '09)*, Honolulu, HI, USA, December 2009.

[71] Roberto Perdisci, Igino Corona, and Giorgio Giacinto. Early detection of malicious flux networks via large-scale passive DNS traffic analysis. *IEEE Transactions on Dependable and Secure Computing*, 9(5):714–726, 2012.

[72] Matthew Prince. The DDoS that almost broke the internet. Published on March 27, 2013. Retrieved on August 1, 2013. http://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet.

[73] Matthew Prince. The DDoS that knocked spamhaus offline (and how we mitigated it). Published on March 20, 2013. Retrieved on August 1, 2013. http://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho.

[74] Matthew B Prince, Benjamin M Dahl, Lee Holloway, Arthur M Keller, and Eric Langheinrich. Understanding how spammers steal your e-mail address: An analysis of the first six months of data from project honey pot. In *Conference on Email and Anti-Spam (CEAS '05)*, Stanford University, July 2005.

[75] Cheng Qia, Xiaojun Chenb, Cui Xud, Jinqiao Shia, and Peipeng Liub. A bigram based real time DNS tunnel detection approach. 17:852–860, May 2013.

[76] Daan Raman, Bjorn De Sutter, Bart Coppens, Stijn Volckaert, Koen De, Pieter Danhieux Bosschere, and Erik Van Buggenhout. DNS tunneling for network penetration. In *Conference on Information Security and Cryptology (ICISC '13)*, volume 7839, pages 55–77. Seoul, Korea, November 2012.

[77] Rod Rasmussen and Paul Vixie. Surveying the DNS threat landscape. Technical report, Internet Identity, 2013. http://www.internetidentity.com/white-papers/.

[78] Michael C. Richardson and DH Redelmeier. Opportunistic encryption using the internet key exchange (IKE). RFC 4322, December 2005.

[79] Rich Rosenbaum. Using the domain name system to store arbitrary string attributes. RFC 1464, May 1993.

[80] William Salusky and Robert Danford. Know your enemy: Fast-flux service networks. http://www.honeynet.org/papers/ff/.

[81] William Yurcik Samuel Patton and David Doss. An achilles heel in signature-based IDS: Squealing false positives in SNORT. In *Symposium on Recent Advances in Intrusion Detection (RAID 01)*, Davis, CA, USA, October 2001.

[82] Ed Skoudis. The six most dangerous new attack techniques and what's coming next? In *RSA Conference (RSA '12)*, San Francisco, CA, USA, February 2012.

[83] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Conference on Computer and communications security (CCS '09)*, Chicago, IL, USA, November 2009.

[84] Tom van Leijenhorst, Darryn Lowe, and KW Chin. On the viability and performance of DNS tunneling. In *Conference on Information Technology and Applications (ICITA '08)*, Cairns, Queensland, Australia, June 2008.

[85] Paul Vixie. Extension mechanisms for DNS (EDNS0). RFC 2671, August 1999.

[86] Paul Vixie and Jun Murai. NCAP - distributed network capture with shared analysis. *Information and Media Technologies*, 6(1):241–251, 2011.

[87] Florian Weimer. Passive DNS replication. In *17th FIRST Conference on Computer Security Incident*, Singapore, June 2005.

[88] Lance Whitney. FBI kills dnschanger network, but how many will be affected? Published on July 9, 2012. Retrieved on August 1, 2013. http://news.cnet.com/8301-1009_3-57468436-83/fbi-kills-dnschanger-network-but-how-many-will-be-affected/.

[89] M. Wong and Wayne Schlitt. Sender policy framework (SPF) for authorizing use of domains in e-mail, version 1. RFC 4408, April 2006.

[90] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Con-*

ference on Internet measurement (IMC '10), Melbourne, Australia, November 2010.

[91] Lotfi Asker Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1978.

[92] Bojan Zdrnja, Nevil Brownlee, and Duane Wessels. Passive monitoring of DNS anomalies. In *Detection of intrusions and malware, and vulnerability assessment (DIMVA '07)*, volume 4579, pages 129–139. Lucerne, Switzerland, July 2007.

Appendix A

Description: A zone file is used to manage the RRs in a nameserver.

```
$ORIGIN example.com.
$TTL 86400
@               IN      SOA    ns1.example.com.      hostmaster.example.com. (
                                2003021106 ; serial
                                10800      ; refresh after 6 hours
                                3600       ; retry after 1 hour
                                604800     ; expire after 1 week
                                86400 )    ; minimum TTL of 1 day
                IN      NS     ns1.example.com.
                IN      NS     ns2.example.com.
ns1             IN      A      10.0.1.12
ns2             IN      A      10.0.1.13
; name          TTL     class  rrtype         rdata
www             3600    IN     A              10.0.1.10
remote          3600    IN     A              10.0.1.11
otherservices   3600    IN     A              10.0.1.15
ftp             3600    IN     CNAME          remote.example.com.
*               3600    IN     CNAME          otherservices.example.com.
;
```

Figure A.1: *Simple Zone File*

Description: The schema of rrset and rrset_reversed tables in our pDNS database project. These tables are used for searching over the rrname field.

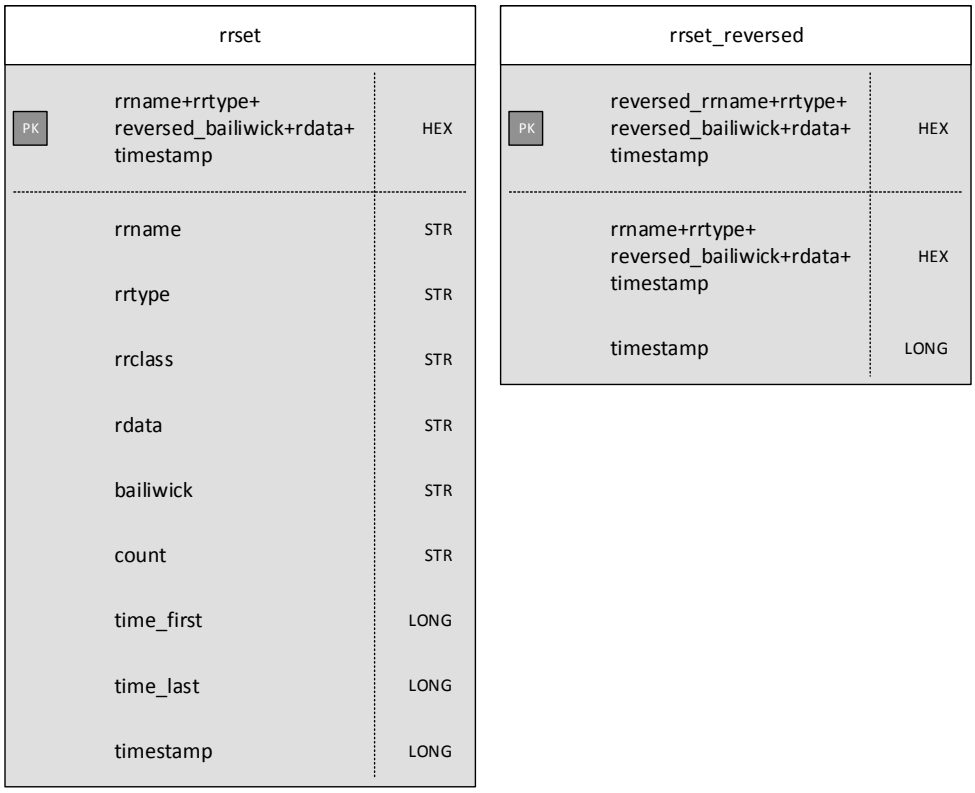


Figure A.2: *The Database Schema: RR Set Tables (+ stands for string concatenation)*

Description: The schema of rdata and rdata_reversed tables in our pDNS database project. These tables are used for searching over the rdata field.

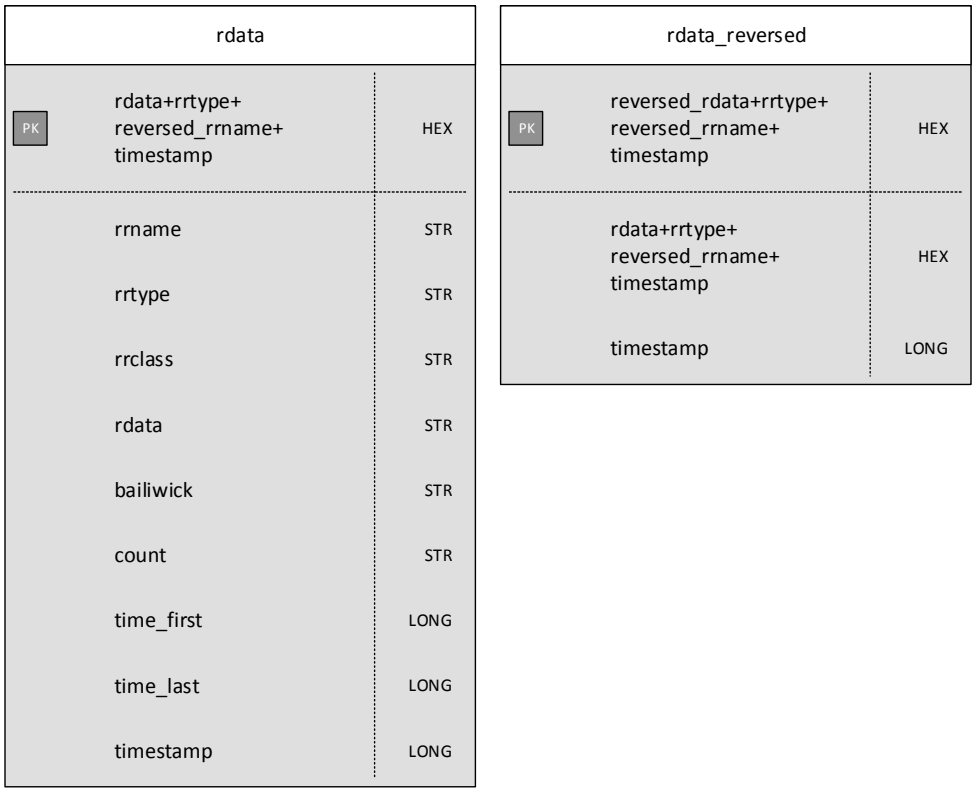


Figure A.3: *The Database Schema: RR Data Tables (+ stands for string concatenation)*

Description: A sample python script for querying our pDNS database. It uses the API of our system through HTTPS, and the result comes in JSON format.

```
import urllib2, json

req = urllib2.Request('https://localhost:8080/?search_type=rrset&\
    rrtype=A&bailiwick=&record_data=*.google.com&limit=3&input_mode=name')
results = list()

req.add_header('Accept', 'application/json')
req.add_header('X-Api-Key', '123456')
http = urllib2.urlopen(req)
while True:
    line = http.readline()
    if not line:
        break
    results.append(json.loads(line))

print results

Result:
[
    [
        {
            u'count': u'90',
            u'time_first': u'2013-08-01 15:28:06',
            u'rrtype': u'A',
            u'rrname': u'news.google.com.',
            u'bailiwick':
            u'google.com.',
            u'rdata': [u'74.125.226.152',u'74.125.226.151',u'74.125.226.159'],
            u'time_last': u'2013-08-01 15:28:06'
        }
    ]
]
```

Figure A.4: *Sample Query by Using the API*