Finding malicious usage via Capture, Storage, Analysis and Visualization of DNS packets

by

Chandan Chowdhury

B. Tech., West Bengal University of Technology, 2007

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2019

Approved by:

Major Professor
Dr. Eugene Vasserman

Copyright

Abstract

The first step of accessing any resource over the internet is to find the IP address of the hosting server corresponding to the easy to remember domain name. For this purpose, the *Domain Name System* (DNS) was introduced in 1985. We can think of DNS as the phone-book directory of the internet. DNS is fundamental to the proper functioning of the modern internet and without DNS it will be very difficult, if not impossible, to navigate the modern internet.

However, like any tool, it is being used for both benign and malicious purposes. Malicious programs use algorithmically generated domains to rendezvous with their command and control server to receive tasks to be performed. DNS has also been used as a covert channel for data exfiltration. Analysis of DNS logs can reveal suspicious domains queried by infected hosts and thus can help prevent and reduce security incidents in a network.

Due to high volume and distributed nature, capturing, logging and analyzing DNS data is non-trivial. In this thesis, we provide a framework for capturing, logging, aggregating and analyzing DNS data and show the results by applying our methods to a university-wide DNS server. We were able to find hosts which were making *Web Proxy Auto Discovery* (WPAD) queries which is used by hosts to automatically find and use web proxy servers available in a network but also makes those hosts vulnerable to *Man-in-the-Middle* (MITM) attack by rogue hosts connected to the same network. The framework also helped us quickly detect and find all hosts in our network which were infected by backdoor put in *CCleaner*, very popular utility software for maintaining Microsoft Windows systems. Investigation of suspicious domains reveals hosts that we believe are running *Potentially Unwanted Programs* (PUP).

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I want to thank my major professor, Dr. Eugene Vasserman for all of his guidance and encouragement over the past years. I also want to thank Dr. Alexandru Bardas for the advice and suggestions on the project, Dalton Hanh for helping in the research and Matthew French for the fantastic web interface that he built on his own. I would like to thank Dr. George Amariucai and Dr. Torben Amtoft for serving on my committee. A big thanks to Kansas State University Networking team and IT Security team for helping us in our research. Thank you Richard Petrie and Qais Tasali for reviewing my thesis. Finally, I want to thank my friends and family for the constant support and encouragement I received.

Dedication

To maa, baba, bhai and all my friends.

Chapter 1

Introduction

The Domain Name System (DNS) is responsible for translating human-readable domain names to Internet protocol (IP) addresses, and vice versa. For instance, when users enter website domain names into their browsers, before anything else, the browser will need the websites IP address to be able to reach it. DNS is used for providing and maintaining this mapping between domains and IP addresses. It is vital for humans to navigate networks large and small, internal and external.

DNS is critical for all types of networks, including the internet, industry, government, academic networks, and the more basic home networks. An attack on a DNS server can be disruptive to all clients dependent on it. For example, a malicious attacker can use *DNS poisoning*[3] to direct users to malicious domains. In October 2016, *Dyn*, a DNS service provider for few of the top web service companies in the USA, was targeted by a Distributed Denial of Service (DDoS) attack which caused those world's top internet services to be unreachable for close to a day in North America and Europe.[4] Similarly, DNS is misused by malware developers: they use *Domain Generation Algorithms* (DGA) to synchronize malware-infected hosts with a remote Command and Control (C2) server and hide the communication in legitimate DNS traffic to avoid detection.[5] *Typo-squatting* is another type of attack, using domain names which looks like a popular domain name and are generally used by malicious attackers to phish for inattentive users.[6]

Since DNS is fundamental to all internet communication, even the security-conscious enterprises allow DNS through their firewalls, making it an attractive channel for attackers. Monitoring and analyzing DNS traffic provides visibility into the communications of a network's internal parties, including malicious communication. The research community has been studying the DNS ecosystem and analyzing DNS traffic for a long time. Work in this area has focused on detecting various security issues, mis-configurations, and misuse/abuse.[3;5;7–9] However, DNS traffic is notoriously difficult to collect and to analyze, especially in enterprise networks. A key challenge is volume, e.g. a typical enterprise network may generate packets in the range of a couple of hundred thousand to billions per day.[10] Even though DNS servers can log DNS traffic themselves, logging is usually turned off as it significantly degrades the performance of the server. Also, even if logging were turned on, most DNS servers log only DNS queries and not the responses. DNS responses may have more security-relevant information than queries and not analyzing DNS responses causes organizations to miss significant opportunities to detect malicious behavior. For example, attackers may hijack a benign domain and point it to an IP address they control. We need to collect DNS responses to be able to detect malicious mapping. Another challenge is distributed hierarchical nature of DNS. While logging at the top level is beneficial, that causes the logger unable to see DNS queries generated by individual hosts behind lower-level DNS servers, for example, departmental DNS servers.

In this thesis, we describe how we built a DNS log analysis framework (supported by a hardware-based physical logger provided by a research partner vendor) to capture and log DNS traffic of a university network and present the result of our analysis of the logs.

1.1 Challenges

Similar to any large network, a US university network is very complex in nature. Generally different groups of networks are distributed over separate buildings, however, one network can also cover multiple buildings. Each separate network is owned by their respective departments and have their own networking equipment and administration teams. Though,

they all connect to a central network which provides the core internet service and connectivity to other departments. Unlike, highly controlled/restricted network, such as corporate, government, a university network is more *"open"* by its nature, due to the fact that different departments has different and occasionally conflicting needs. Thus, a single university-wide security policy is not feasible. For example, while individual departments can have their DNS records in university's central DNS server, for flexibility, they can choose to run their own DNS server using their own preferred DNS server software, such as Microsoft DNS server, BIND, NSD. Similar situations can be found regarding networking equipment and servers where individual departments use their own preferred hardware and software.

This diverse set of networks and hosts makes it difficult to use a common set of commercial or free tools to monitor all hosts running inside the university network. Our framework provides a complementary service to the central networking and security team which uses their own tools, e.g. firewalls, log monitoring, deep packet inspection. For example, although the network firewall will stop malicious traffic on well-known ports, it most likely will not stop the malicious communication if DNS is used as a covert channel.

1.2 Background

DNS follows a hierarchical naming structure separated by periods. Let us use `www.example.com` as a domain name.[11] In this case, `com` is called the *Top Level Domain* (TLD) and `example` is the second level domain. The TLD categorizes the domains in type or location while the first level domain identifies individual entities such as organizations, companies, persons.

As of February 1, 2019, there are 1,535 top level domains[12] and more are being created every year. A few of the most well-known TLDs are `com`, `org`, `edu`, and `net`. These are called *"generic"* Top Level Domain or gTLD. Then there are top-level domains specific to countries based on *"Country Code"*, known as ccTLD. All these top-level domains belong to what is called the *"root zone"*. Logically there are thirteen *"Root Servers"* operated by twelve *"Root Server Operators"* such as VeriSign, ICANN, and RIPE.[1] As of February 1, 2019, there are 933 instances of different root servers running all over the world[1] with the

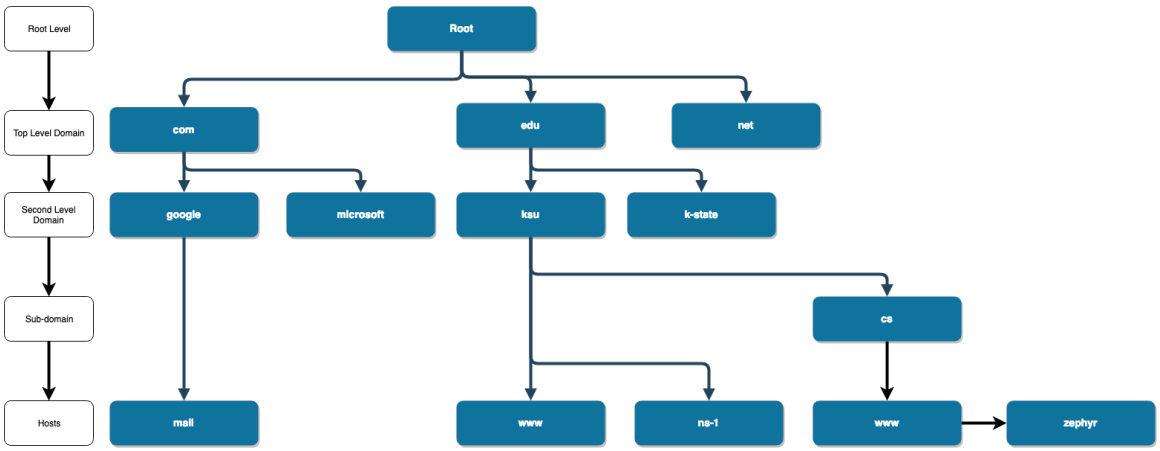


Figure 1.1: *DNS Hierarchy*

number generally increasing every year. For example, as of February 19, 2019, ICAAN, together with its partners, runs 163 instances of L-root server spread over in 82 countries.[13]

Root Server	Root Server Operator
a.root-servers.org	Verisign, Inc.
b.root-servers.org	University of Southern California (Information Sciences Institute)
c.root-servers.org	Cogent Communications
d.root-servers.org	University of Maryland
e.root-servers.org	NASA (Ames Research Center)
f.root-servers.org	Internet Systems Consortium, Inc.
g.root-servers.org	US Department of Defense (Network Information Center)
h.root-servers.org	US Army (Research Lab)
i.root-servers.org	Netnod
j.root-servers.org	Verisign, Inc.
k.root-servers.org	RIPE NCC
l.root-servers.org	ICANN
m.root-servers.org	WIDE Project

Table 1.1: *Root Servers Home Page and Operators.*[1]

DNS generally works over *User Datagram Protocol* (UDP) but can be configured to work over *Transmission Control Protocol* (TCP). [1] When a host connects to a network, the *Dynamic Host Configuration Protocol* (DHCP) server of the network which is responsible for allocating IP addresses in the network, assigns a unique network IP address to the host and provides a list of DNS servers to query by default. To resolve a domain name, the request goes from a program running in the host to the local *"DNS Resolver"* which is generally part

[1] Administrative operations like "Zone Transfer" are performed over TCP for reliability purposes.

of the host operating system. If the host-local DNS resolver does not already have the IP address corresponding to the domain, it sends a *"DNS Query"* packet to the first DNS server on the list of DNS servers provided by the DHCP server. The DNS server, upon receiving the query, searches its own database for the domain name. If the domain details are found then DNS server returns one or more *"DNS Response"* packets to the requesting host.

If the default DNS server does not have the details of the domain in its own database, two situations can occur based on how the default DNS server has been configured. A DNS server can be configured in *"Recursive"* or *"Non-recursive"* mode.

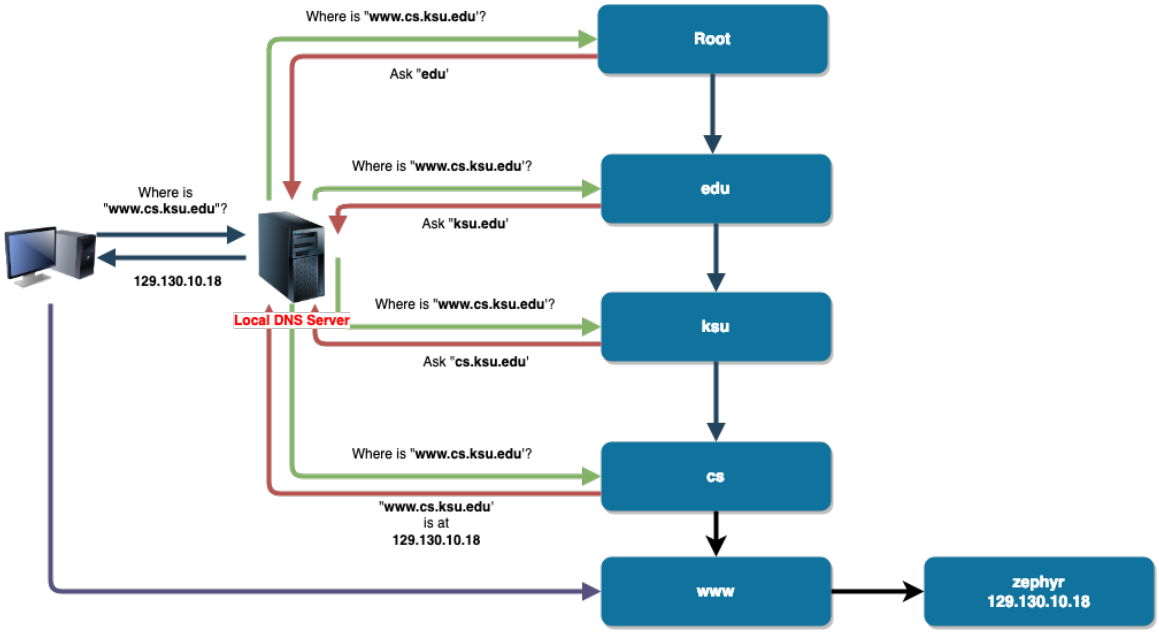


Figure 1.2: *DNS Recursive mode*

When a DNS server has been configured in recursive mode and the requested domain is not found in its own database, the DNS server will query other DNS servers upwards in its hierarchy with which it has been configured, which may in turn query other servers, and so on. Eventually, the recursive server will be the one to return the final result to the host which requested the domain. The recursive DNS server also keeps a copy of the new found DNS record in its own database ("cache") to expedite future name resolution by same or other clients. The duration for which the record is kept depends on the *"Time To Live"* (TTL) parameter set in the DNS record. TTL is expressed in seconds and indicates the minimum

time during which the record will not be changed. For example, a TTL value of "86400" indicates that a record should be valid at least for 24 hours. So if a client has a DNS record which was fetched more than 24 hours earlier, it should query for the DNS record again to make sure it has the updated version. A lower TTL value will make clients query for DNS record more often thus increasing the load on the DNS server but allowing changes to DNS record spread relatively quickly. A higher TTL value has exactly the opposite effect i.e. less load on DNS servers but also a longer waiting period before changes start propagating.

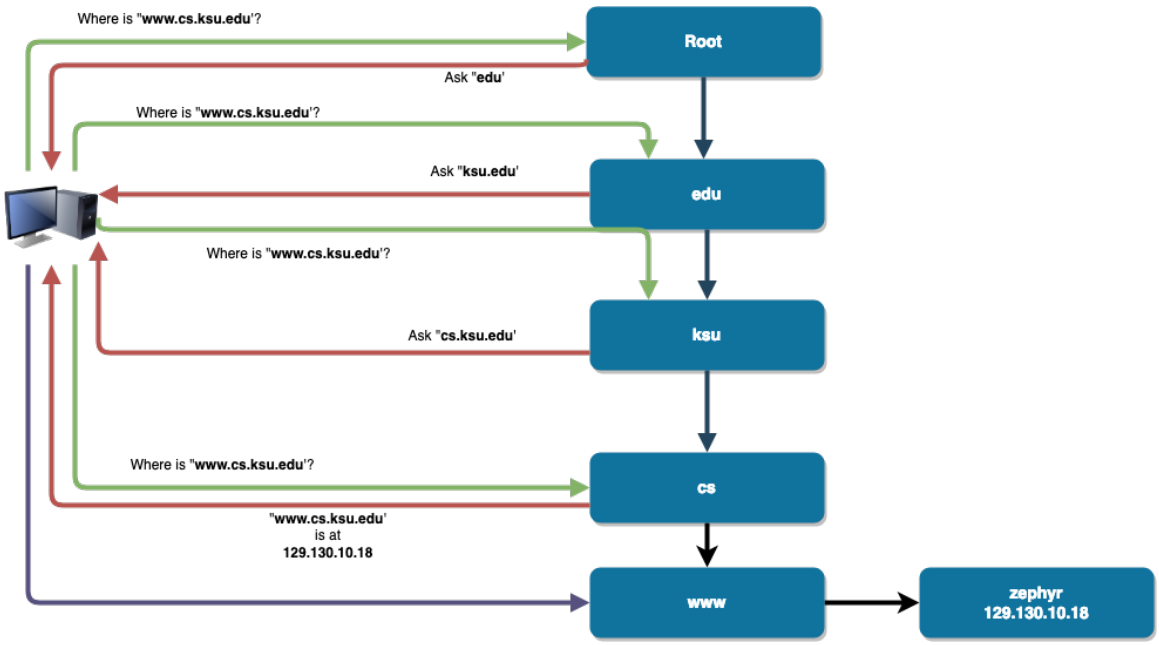


Figure 1.3: *DNS Iterative mode*

When a DNS server is configured in non-recursive mode, it does not resolve domains by querying other DNS servers upward in its hierarchy. Rather, it would find the *"authoritative"* DNS server for the top-level domain, which keeps the record of all domains under it and replies the host-local resolver with that information. The top-level authoritative DNS server keeps a database of all authoritative DNS servers for all second-level domains. Similarly, a second-level authoritative DNS server keeps the record of all sub-domains and hosts under it. The host-local DNS resolver will first query the authoritative DNS server for the top-level and ask for details about the second-level domain. So in case of `www.example.com` above, the hosts DNS server will find the authoritative DNS server for `com` and will inform the

host-local DNS resolver to query it for `example.com`. When the authoritative DNS server for `com` receives the query for `example.com`, it will look up its own database to find the authoritative DNS server for `example.com` if it exists. In case the server cannot find the authoritative DNS server for `example.com` in its database, it will return a record to the host-local DNS resolver indicating that the domain does not exist (NXDOMAIN). If the authoritative DNS server for `example.com` is found, the host-local DNS resolver will next query it for `www.example.com`. If `www` exists in the database of the authoritative DNS server for `example.com`, it will return the DNS record, or else it will return NXDOMAIN record indicating `example.com` does not contain `www`.

All transactions between a host and a DNS server occurs using various types of DNS records. The host sends a *"Query"* record and the DNS server replies with a *"Response"* record. The details stored by a DNS server are known as *"Resource Records"*. A part of the resource record called *"Header"* is fixed while rest is dynamic. The header contains details such as Type of the record (query or response), Operation code, Response code, count of each type of records included in the packet. The dynamic part contains *"record type"*, *"record length"* and then the actual record content.

When querying a DNS server, generally a host asks for the IP address corresponding to a domain, which is called an "A" type record. However, DNS can and does store many other types of record, such as "SOA", "AAAA", "PTR", "MAIL", "TXT". Most of those records are for computers which can look up those records and take action. For example, TXT records were originally introduced for arbitrary human-readable text data regarding a host or domain. In 1993, RFC-1464[14] was introduced to standardize the content to be used for various purpose. For example, to prove that a user owns a domain, well-known services like Google and Facebook can instruct that user to set a custom TXT records in the domains DNS data with content provided by the service company.[15;16] The service company periodically tries to find the TXT record with the content supplied by them. If the specific TXT record is found, the user is considered as an owner of the domain. *"Sender Policy Framework"* (SPF)[17] is another use of TXT record where a domain informs the world that any email originating from that domain will always be sent from IP(s) mentioned in the SPF

TXT record. When another email server receives an email with a domain as sender but not sent from the IP(s) mentioned in the sending domains SPF record, the receiving email server generally mark the email as spoofed and take action such as reject the email completely or move it to a spam folder, for example. *"Domain-based Message Authentication, Reporting, and Conformance"* (DMARC)[18] takes SPF a step further and using TXT record informs the receiving email server what action to take when an email is received from the sender domain but not from the IP(s) mentioned in the senders SPF record. For example, the receiver email server can report the email to the sender domain via a field mentioned in DMARC record or the receiver domain can send an aggregate report to the sender domain.

Once a DNS server has processed query from a host, it will create and send a reply to inform the host of the result. The first thing the reply from the DNS server contains is a 4-bit *"Response code"* based on the result of the domain name search. The codes can be one of NOERROR, FORMERR, SERVFAIL, NXDOMAIN, REFUSED, YXDOMAIN, etc. Table 1.2 shows common codes and their meaning. A comprehensive list can be found in RFC-6895.[19] Every response code has its own format of record called *"Response Record"*. Each response record contains more details about the domain name search and its output. The reply can contains other type of records such as "NS" (authoritative name server), "A" (host IPv4 address), "AAAA" (host IPv6 address), "MX" (mail server address), "TXT" (data in text string format) corresponding to a domain.

Flag Value	Response Code	Meaning
0	NOERROR	Valid query
1	FORMERR	Name server was unable to interpret the query
2	SERVFAIL	Name server internal failure
3	NXDOMAIN	Domain name does not exists
4	NOTIMP	Query type not supported
5	REFUSED	Name server refuses to perform the requested operation
6	YXDOMAIN	Domain name should not exists but found
9	NOTAUTH	Name server not authoritative

Table 1.2: *Common DNS Response Codes and their meaning*[2]

1.3 Related work

Due to the critical role that DNS plays in public internet, the research community has studied DNS extensively, generating excellent findings and data. The goal of this thesis is to combine the finds into a single framework, making it a single point of source to report on the health of a network from the perspective of DNS.

Antonakakis et. al. has developed Notos[7] to identify malicious domains from benign ones based on characteristics such as syntactic pattern, registration information, or access pattern. However, the research was heavily dependent on already available malicious domain data.

Others[9;20–26] have taken a different approach of finding malicious infections using only the DNS traffic data. Their research shows that malicious programs which use DNS for communication will have a distinct signature in DNS traffic. They applied diverse techniques including but not limited to lexical analysis, data mining, graphical inference, and/or machine learning to find the signature. Choi et. al.[20] has identified 14 features of DNS data to detect malicious botnets in a network. They found that in most cases a malicious domain name will contain either well-known domain names such as Microsoft, Windows, Gmail, Facebook or well-known services such as mail, update, news. They also found that most of the time malicious domains will resolve to IP of the host itself or to a private IP range which effectively causes the malware to cease communication to avoid detection.

Stalmans et. al.[27] placed a system at the edge of a large South African university to monitor DNS activity. The authors used various statistical methods to detect malicious domains which are using domain ‘‘fast-fluxing” which is a method where the IP address corresponding to a domain is changed often so that when one IP is compromised a new IP is used for communication making it very difficult to detect and block the communication. Their system uses statistical measures and was able to detect malicious domains with a high degree of accuracy, minimizing the need for blacklists. Thomas & Mohaisen[28] analyzed DNS traffic of a top-level authoritative domain name server. Specifically, they monitored `com`, `net`, `tv` and `cc` top-level domains. They found that unique NXDomains (see 1.2) are typically

observed more than once within a week and only a small percentage reoccur over multiple days. They also observe that the *Domain Generation Algorithm* (DGA) generated domains receives significant NXDomains one day prior and post specific generation date. They believe this may be a *"side effect of global clock skew"*. When malware is using algorithms to generate domain names of their command and control server, the majority of the time the generated domain will not exist. Plohmann et al.[5] and Yadav & Reddy[25], exploited this approach to find malicious DNS traffic. Perdisci et. al.[29] analyzed 2.5 billion DNS queries per day for 45 days from two large Internet Service Provider (ISP) networks. They show us how to use passive analysis of recursive DNS traffic traces to find malicious flux service networks. They claim that their approach was able to detect malicious domains "in-the-wild" i.e. those domains which are malicious but has not yet been categorized as malicious. They believe their work should help improve spam filtering by automatically detecting and filtering emails containing links to malicious domains. On the other hand, Chang et. al.[30] uses text categorization of DNS queries to fingerprint the underlying operating system (OS) and uses the findings to detect if NAT/tethering is being used. They use domain names queried by OS, for example, query for OS updates, as a detection feature. They then categorize the found domain names to the OS detected by DHCP server. Their research assumes that DHCP detects the device as having the correct OS and uses it as ground truth, but later in the paper, they agree that the DHCP data may be imprecise. They conclude, that queries from single host appearing under more than one OS category is most likely the result of more than one OS being run behind the same IP and thus NAT/tethering is being used. They claim that their data "reveals that although most mobile devices has tethering function that allows them to share the internet connection with laptops, people prefer to use notebook devices as hotspot". However, in their paper, they never mentioned if they had obtained and correlated the DNS data from users mobile devices which only the ISP of the cell service used by the mobile device will have.

Chapter 2

Architecture

As shown in Fig-2.1, our campus has three main central DNS servers serving different sections of the campus. We are monitoring the Central Main DNS Server-3. By default, the central servers serve DNS requests from departmental servers and wireless clients. To balance the load, the DHCP configuration for each subnet distributes the hosts among the central DNS servers.

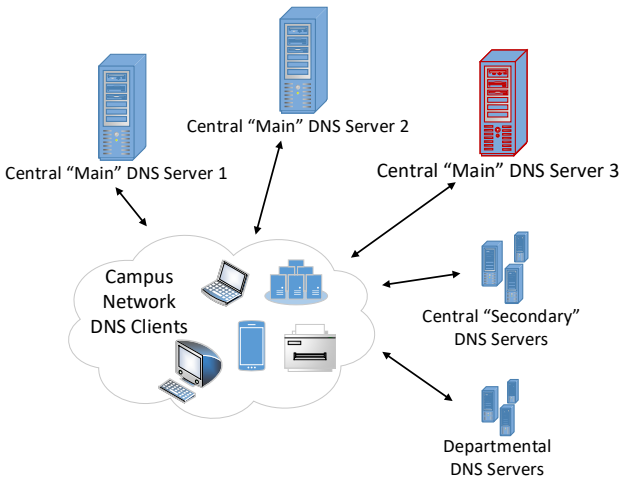


Figure 2.1: *Campus DNS Infrastructure Overview – The infrastructure is highly distributed with servers located in different data centers. We are monitoring Central "Main" DNS Server 3.*[2]

[2]Reproduced with permission from[31]

Inspired by the always watching *"eye of Sauron"* from the *"Lord of the Rings"* trilogy, we call our framework *"eyeDNS"*. The framework has three main components: DNS Logger Appliance, Data Processing (backend), and Visualization (frontend). Figure-2.2 provides and overview of the framework and it's components. This separation allows the software developers to handle issues and make changes in each component individually while other components continue to function. For example, when the *Logger* is turned off due to software or OS update, the data processing and visualization component will still be available. Similarly, when we are making changes to the visualization component, the logger and data processing component is not affected, and continue to log and process data. Below we give details about each component.

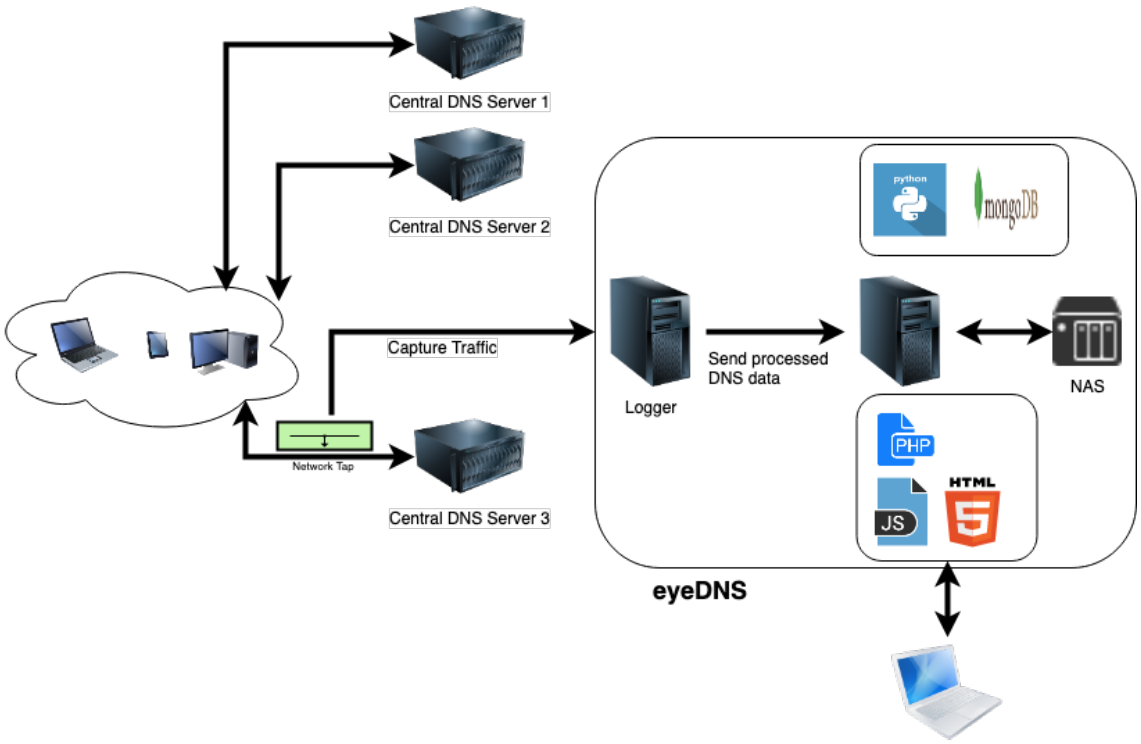


Figure 2.2: *eyeDNS Framework and Components*

2.1 DNS Logger Appliance

The purpose of the DNS Logger is to passively watch and record DNS traffic on the traffic. The Logger appliance hardware was provided to us by a research vendor. It is a Microsoft Windows 2008 Server host with a high-performance network card and a specialized proprietary program. As explained in Section-2, the appliance is connected to one of the three main departmental servers via a *"tap"* connection. The tap mirrors all traffic going in and out of the DNS server and sends the mirrored traffic to the Logger. The high-performance network card in the Logger captures the traffic and filter out everything other than DNS traffic. The captured DNS traffic is then read and analyzed by the specialized proprietary program. The program records and aggregates DNS data based on IP address, domain name, time-period, etc. Part of the program behavior can be altered via parameters supplied through a configuration file. For example, the time period after which the program must write analyzed data to file is set using a parameter in the configuration file. We can modify the configuration file to change this time period to save the data more often or less often. While saving more often will reduce the chance of losing the analyzed data in case the Logger or the OS crashes, saving less often creates less number of files. Currently, the analyzed data is saved every hour in a new set of files.

The Logger has the capability to read a list of domain types and categorize the queries and responses before saving them in files. The domains can be categorized into two lists a) *"Whitelist"* and b) *"Blacklist"*. Domains which are known to be benign are part of the *"whitelist"*, while known malicious domains are part of the *"blacklist"*. Domains that are not part of the whitelist or the blacklist falls into *graylist*. The domains in the blacklist list file can also have an associated score indicating the confidence of the maliciousness of the domain. For example, a score of 100 indicates a domain is definitely malicious, where a score of 50 indicates that there is a 50% chance that the domain is malicious. Using the Black list score along with the number of queries made by a host to a malicious domain, the Logger can calculate an overall score for a host which is an indicator of malicious behavior performed by the host. During the initial phase of our research we were provided with a

commercial blacklist file by our vendor, however, that is not that case anymore and the eyeDNS framework is not dependent on a blacklist.

The data capture files generated by the logger are

1. Overall statistics of various data points, which include, the total number of DNS queries and responses, the total number of NXDomain responses, Number of queries and responses from whitelist, blacklist and graylisted domains, type of DNS records seen and their count, etc.

2. List of whitelist, blacklist and graylist domains queried and how many times.

3. Statistics for a single IP address which include details like total query count, number of queries with NXDomain response.

4. Domains queried along with the list of IP addresses and number of times the host queried the domain.

5. List of invalid domains queried with counts.

6. NXDomain statistics.

2.1.1 Data Transfer

One of the challenges we faced was the automatic secure transfer of files from the Windows-based logger to Linux based data processing host. Unlike most Linux based systems, Windows does not come out of the box with any secure file transfer clients, such as *Secure Copy* (SCP) or *Secure File Transfer Protocol* (SFTP) clients. There are both commercial and free software available e.g. PuTTY/PSCP, Filezilla, WinSCP which can be installed and used for secure file transfer. We had two criteria; first, the tool must support secure file transfer protocols such as SCP or SFTP and second, must be able to be scripted so that it can transfer files without any user interaction. After evaluating available options, we found "WinSCP" with scripting mode to be most suitable for our purpose. WinSCP scripts allow various customization option including filtering files based on multiple different parameters

like file name prefix, type, different timestamps; specifying destination host, port, credential, etc. Using the available parameters we created a WinSCP script which filters files from a source directory using file name prefix and type, login to our data processing back-end server using SCP and transfers the files. We provide a sample script in Appendix-A.

Next, we created a Windows scheduled task to run 3 minutes past every hour which follows the schedule of the logger which generates the files at 1 minute past every hour. The Windows task invokes a batch program which in turn executes the WinSCP script to securely transfer files to the destination. Once the transfer is completed successfully, the files are moved into a backup directory.

The whole process is accommodating of any delay/change in the schedule. The script and scheduled task run in such a way that if any number of schedule is missed due to downtime or any unforeseeable circumstance, the next schedule recovers the whole process. Thus we can have the data processing back-end servers down for regular maintenance or unplanned events, and yet when the servers come back online, all files that should have been sent earlier is sent altogether for processing. From there, the data processing back-end server resumes the data processing as if no delay has happened.

2.2 Data Processing

Once the log files from the Logger are sent securely to our data processing host (Section-2.1.1), they need to be processed and aggregated further to get more information out of them. Every hour, a scheduled job kicks in and executes a set of Python scripts to process the log files. The overall steps of log processing follow the steps described in Algorithm 1.

The secure file transfer process puts the files in a *Network File Storage* (NFS) shared directory which is only accessible from a restricted set of hosts such as our data processing host. At 5 minutes past every hour, a *cron* (which is a task scheduler for Unix-like operating systems) job kicks in and executes a set of python scripts.

The scripts are functionally divided into two parts. The first set reads the files and directly insert the records in MongoDB (Section-2.2.1) collection. The second set, read the

```
The files are received in a specific directory;
while File Exists do
    Production jobs processes the file;
    if Production processing is successful then
        Move the file to development process directory;
        Development jobs processes the same files;
        if Development processing is successful then
            Move the files to backup directory;
        end
    end
end
```

Algorithm 1: Log file processing stages

inserted records and generate aggregate records which help in calculating various statistic about the data.

The development jobs follow the same steps but with development code-base instead of production. This setup makes it easier to test new code changes without impacting production environment. New features are pushed into the development branch of code-base. After all the tests are passed, we manually verify the data to make sure the changes have intended impact and most importantly we do not have any unintended impact. Once everything checks out as they should be, the changes are pulled into production code-base.

2.2.1 MongoDB

As explained in Section-2.2, processed records and aggregates are stored in a MongoDB database. MongoDB is a *"Document Store"* based NoSQL database server which stores a record as a *"document"*. Documents of the same type are organized into a *"collection"*. One of the major features of NoSQL databases is that it is *"Schema-less"*, which means we can store documents with different number and types of fields in the same collection as long as all documents in the collection have common field(s) to uniquely identify each document; this is not possible with *"tables"* used in *Relational Database Management Systems* (RDBMS). The "Schema-less" feature allows adding new fields to a record or removing existing field while developing programs which in turn reduces the overhead of needing to destroy and

recreate the whole table which is required in case of traditional SQL databases such as RDBMS. MongoDB is also optimized for storage and fast retrieval of *JavaScript Object Notation* (JSON) documents which we used to store our records. The disadvantage of a NoSQL database is they do not support direct joins on different databases making fetching related records from other databases in a single query difficult. However, this lack of direct join by a single query is overcome by performing multiple queries and then joining related records in the program. Another short-coming, a field name of a JSON document can be larger than the actual value it is holding, and because JSON documents must also store all field names along with their value, it increases the size of each record and thus increases storage cost compared to RDBMS. However, with the cost of storage decreasing every year and having an NFS based data storage device, the cost of storage has never been a concern for us.

2.3 Visualization

The Visualization component is a very important part of the eyeDNS framework and is used to view the processed data in various forms including graphical, tabular, etc. The visualization component is composed of two parts. The first part is a set of PHP programs running on Apache web server. The PHP programs provide a set of *Web Services* for querying and fetching data from the MongoDB server. The second part is a custom developed JavaScript based front-end web application also running on the same Apache web server. The front-end displays the data received by querying the REST API in various visual forms. To be able to access the JavaScript front-end, a user must be authenticated first. We have used mod_auth_cas[32] which is a plugin for Apache web server and is used for configuring the Apache web server to use Single-Sign-On (SSO) based authentication. The advantage of the SSO based authentication is that we do not have to manage user credentials. When a user wants to access the eyeDNS front-end, the web server first sends the user to the pre-configured SSO service provider which authenticates the user and redirects back to the eyeDNS front-end once the authentication process completes successfully. The authentication

process follows Algorithm 2.

```
User open their browser and enter eyeDNS URL;
Request is received by our Apache web server;
Apache web server checks authentication cookie;
while authentication cookie is not present or invalid do
    Apache uses mod_auth_cas configuration to find the login URL;
    Setup callback URL in request to return the user back to the eyeDNS page when
     login is successful;
    Redirects the users browser to the central SSO login page;
    while User not validated by Central SSO Server do
        Central server asks for username and password to authenticate the user;
        if Successfully authenticated then
            Central server set the authentication cookie;
            Redirects the user to the callback URL provided in the redirection request;
        end
    end
    Apache web server calls mod_auth_cas to validate the authentication cookie;
end
User is allowed to access eyeDNS frontend;
```

Algorithm 2: User authentication

The JavaScript front-end application is connected to the PHP based back-end via Web Services. The PHP programs expose a set of *Representational State Transfer* (REST) *Application Programming Interface* (API). In simple terms, each API is a web URL with parameters and returns data in JSON format to the caller. This separation allows the same REST API to be used for accessing the data via other tools. For example, the REST API can be used by a python program which can run machine learning algorithms. The front-end is used to view daily graphs and tables displaying the hourly values of overall statistics, domains queried by a single IP in a period, most queried domains in a period, etc.

After a user is authenticated, they can use one of the available tabs (2.3) to view different statistics. In most cases, user interaction follows a pattern. The user clicks on a specific tab corresponding to the type of statistic she/he is interested in, enters required details for that tab such as date range, IP or domain and finally click submit.

[3]Sauron logo property of Middle-earth Enterprises
[4]Reproduced with permission from[31]

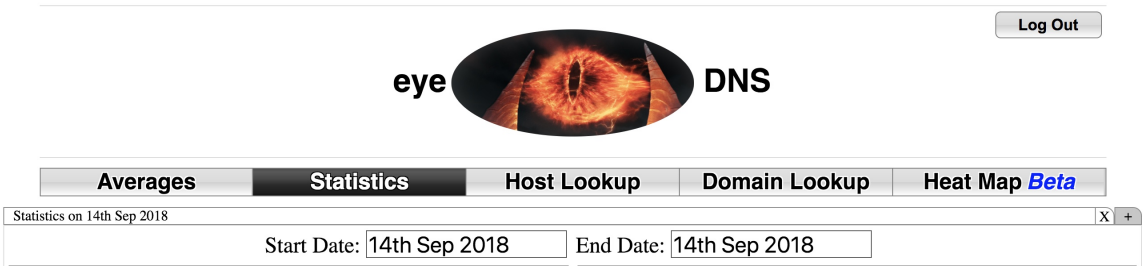


Figure 2.3: *Screenshot of tabs in eyeDNS web interface*[3]

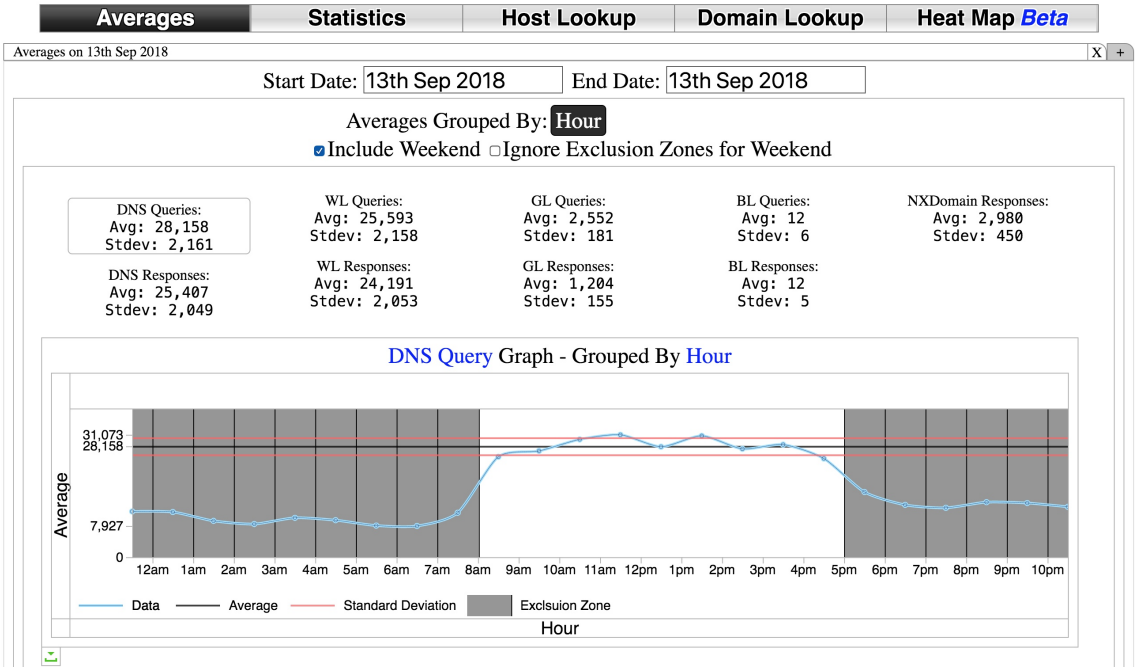


Figure 2.4: *Screenshot of Averages tab in eyeDNS web interface*[4]

In the first tab, Averages (2.4), users specify a date range and the *User Interface* (UI) displays average and standard deviation data and graph for different DNS traffic between the specified date range. For example, we can get the average and standard deviation of DNS queries and responses of the types of domains (whitelist, graylist, and blacklist), NXDomain. Based on the number of days specified, the data and graph can be grouped by hour, day, week or month. Users can include or exclude weekends in case they are only interested in average during weekdays which is Monday through Friday.

The second tab, Statistics (2.5), accepts a date range and displays data for hourly DNS

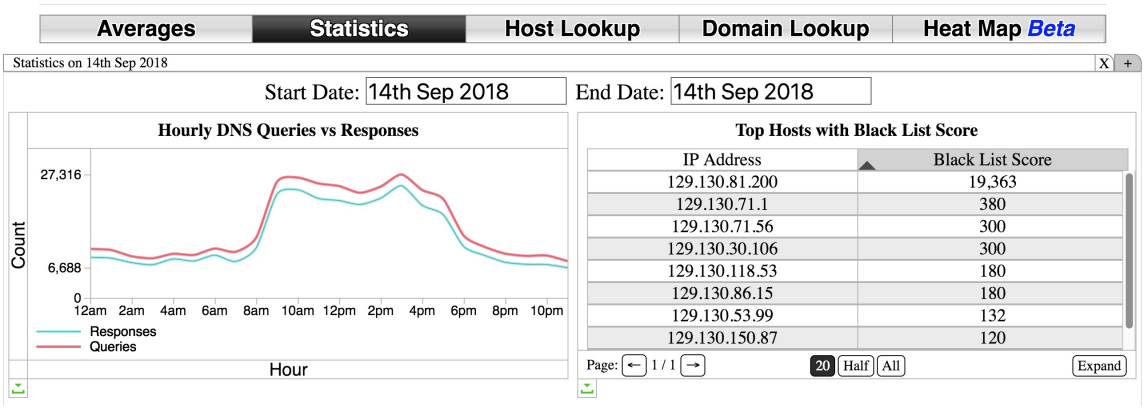


Figure 2.5: *Screenshot of Statistics tab in eyeDNS web interface*

query and response counts for different types of domains, NXDomain, queries generating NXDomain response, top hosts with the highest blacklist and graylist score, top queried blacklist and graylist domains, invalid domains queried, first sighted domains, etc.

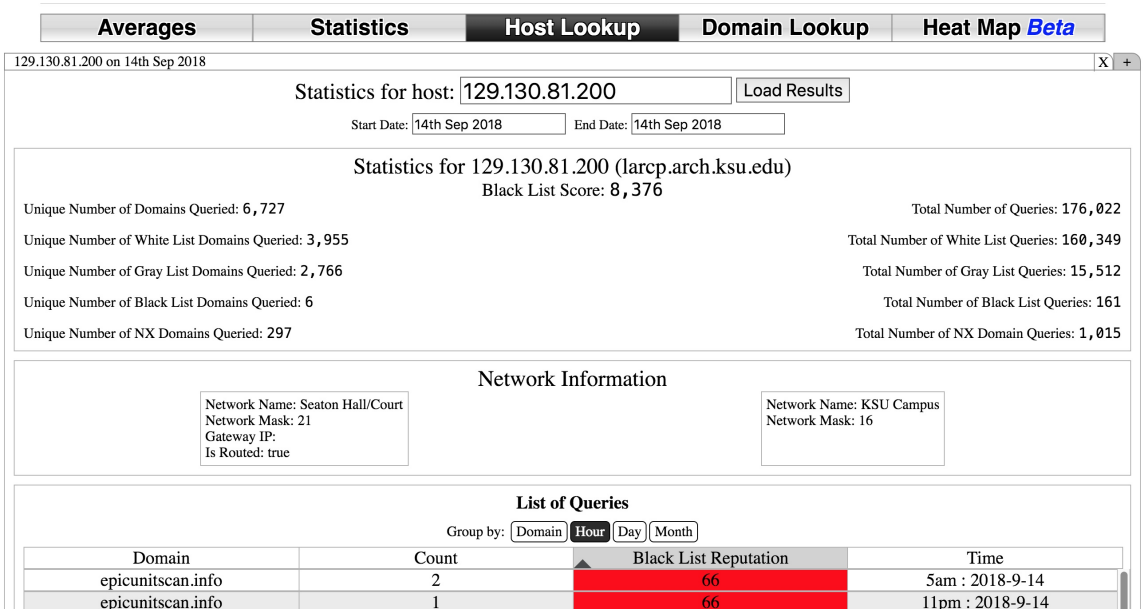


Figure 2.6: *Screenshot of Host Lookup tab in the eyeDNS web interface*

In the Host Lookup tab (2.6), users enter a host IP and date range to find data about all the domains the host has queried in that date range. The data include which domains were queried by this host, during which hour and how many time in that hour. The queries can be grouped by domain, hour, day or month. The tab also shows data about domains

queried by the host which caused NXDomain response.

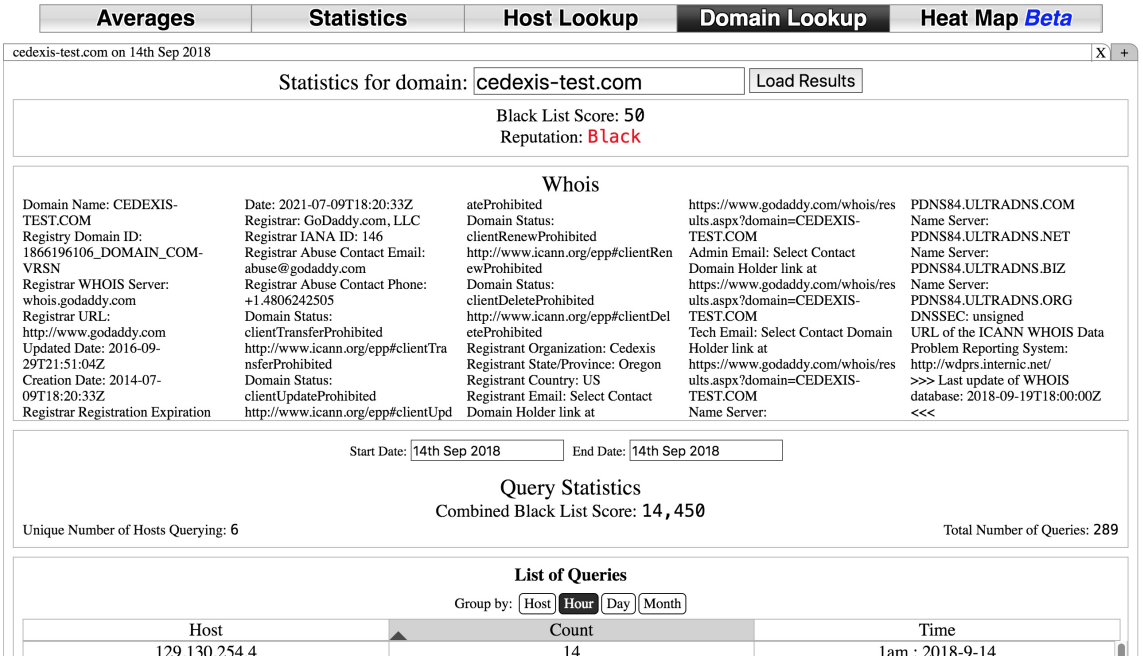


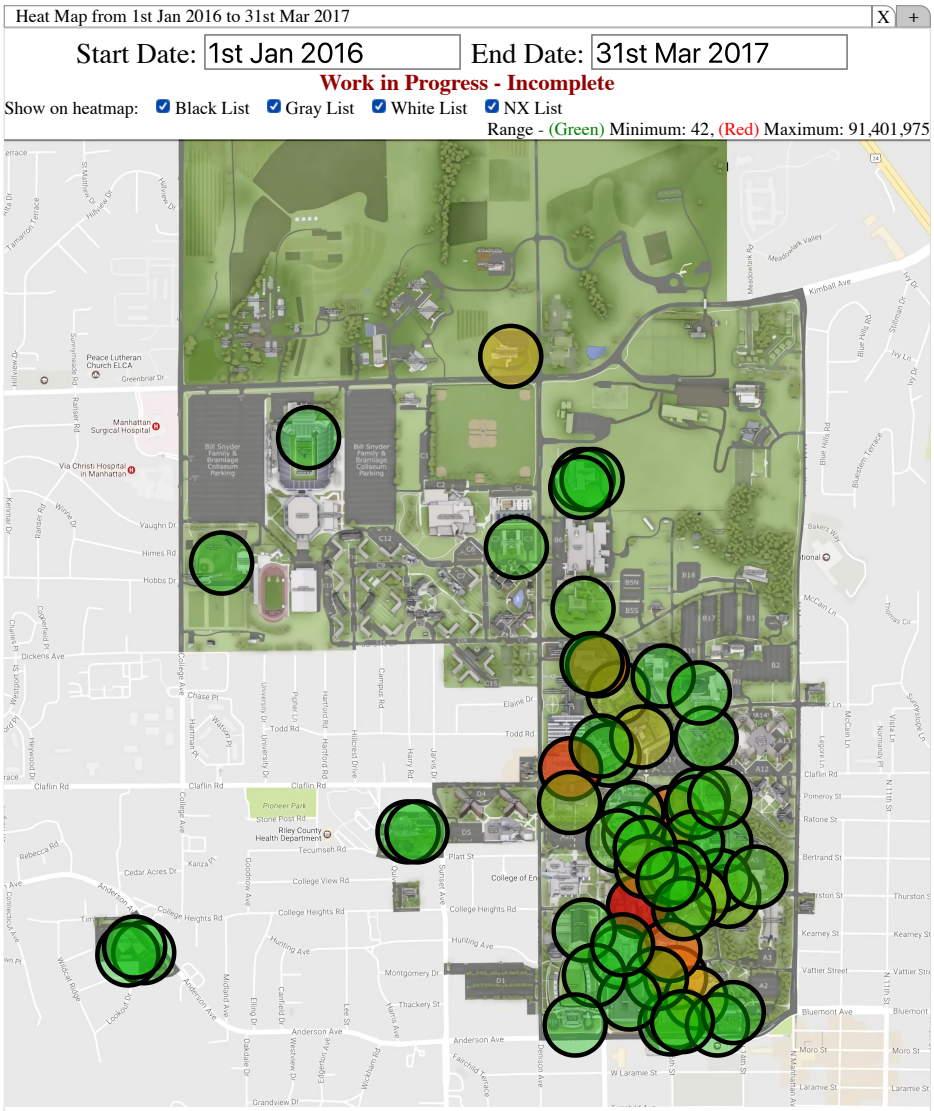
Figure 2.7: *Screenshot of Domain Lookup tabs in the eyeDNS web interface*

The Domain Lookup tab (2.7) is very similar to the Host Lookup tab and shows data about a domain instead. The data includes reputation information, "whois" information, the total count of queries for this domain by each host in an hour during a specified date range. The data can be grouped by host, hour, day, week or month.

The last tab Heatmap (2.8) show a visual representation of counts of DNS, BL/WL/GL queries, NXDomain responses over a campus map. The heatmap is divided into buildings based on network segments and helps visualize the health of network segments. For example, a network segment with a red circle indicates poor DNS health caused by more black-listed domain queries compared to other segments. Similarly, more NXDomain responses will also cause the network segment to be highlighted as the high number of NXDomain responses can be contributed to malware trying to reach their Command and Control server.[5;25]

All of the above-explained web UI tabs are generated by custom JavaScript code. The JavaScript code accepts input from a user and calls corresponding REST API along with other required and/or optional parameters as URL encoded key-value pair. The PHP scripts

behind the REST API read the JavaScript provided inputs to build filter queries to extract data from various MongoDB collections. The PHP scripts process the retrieved data as per the request, formats the processing output in JSON and return it to JavaScript to be displayed in the web UI in various forms.



List of Building Query Counts (Highlighted rows aren't shown on the Heat Map)

Building	Total Queries	Black List Queries	Gray List Queries	White List Queries	NX Queries
Off-Campus or Private Range	381,149,013	536,079	19,693,770	360,919,164	20,571,172
Seaton Hall/Court	91,401,975	110,188	5,985,599	85,306,188	877,903
Throckmorton Hall	77,832,038	164,374	3,941,032	73,726,632	5,045,364

Figure 2.8: *Screenshot of Heatmap tabs in the eyeDNS web interface*

Chapter 3

Results and Findings

By analyzing the data we were able to answer various questions about the DNS traffic of our campus. For example, what is the average number of DNS query and response per hour during a weekday? What is the standard deviation of the same? How many NXDomain responses were received each hour? What are some of the top queried domains?

In the following sections, we present the results we were able to find using our framework. Before we begin, we should point out that as per our central networking team the Central DNS Server 3 which we monitored handles around 15% of total university DNS traffic, which means all the counts we show corresponding to 15% of total university DNS traffic.

3.1 Overall stats for 2017

We have collected and analyzed DNS queries and responses over a 15 months period, starting from January 1, 2016, to March 31, 2017. In total we had 640,891,500 DNS queries and 696,879,237 DNS responses.

As shown in Table 3.1, we had on average 131,159 DNS queries and 140,994 DNS responses per hour during weekday business hours. During non-business hours, the per hour average drops to 32,748 queries and 36,215 responses which are around 4 times less compared to business hours. When combined together we had 59,256 queries and 64,439 responses on

average per hour for weekdays.

January 1, 2016 - March 31, 2017						
	Business Hours (8am - 5pm)		Non-Business Hours		Overall	
	Hourly avg.	stdev	Hourly avg.	stdev	Hourly avg.	stdev
DNS Query Packets	131,159	46,912	32,748	16,721	59,256	51,992
DNS Response Packets	140,994	46,957	36,215	17,028	64,439	54,465

Table 3.1: *Hourly DNS Traffic Averages – A comparison between the hourly averages of DNS packets processed during "business hours" vs. "non-business hours." Most of the domains encountered in the DNS resource records are on the whitelist (Alexa top 1 million sites)*[2]

Figure 3.1 plots the hourly counts in graphical format. The trend follows the data shown in Table 3.1 which shows that during working hours, which is 8 AM to 5 PM, the counts are highest compared to non-working hours.

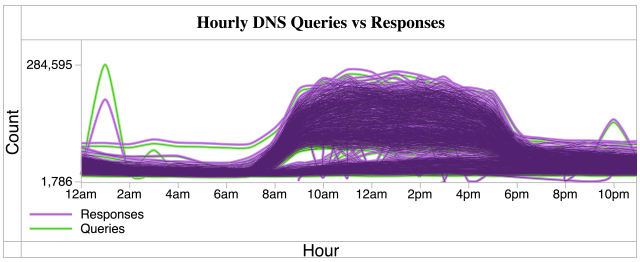


Figure 3.1: *Overall distribution of all the queries and responses from January 1, 2016, to March 31, 2017. The workday is clearly visible from 8 am to 6 pm. Note the outlier: 12 am to 2 am.*[4]

A similar trend can be seen in Table 3.2 between weekday and weekend. While daily average was 1,714,726 DNS queries and 1,857,962 DNS responses per day on a weekday, the average numbers drop around 2.5 times during weekend to 677,358 queries and 753,533 responses.

Table 3.3 shows average DNS queries and responses per semester for Spring, Summer, and Fall. As we can see Fall had the highest number of average queries and responses, followed by Spring and Summer. The findings correspond to the facts that our university has more

[2]Reproduced with permission from[31]
[4]Reproduced with permission from[31]
[6]Reproduced with permission from[31]

January 1, 2016 - March 31, 2017						
	Weekdays		Weekends (Saturday and Sunday)		Overall	
	Daily avg.	stdev	Daily avg.	stdev	Daily avg.	stdev
DNS Query Packets	1,714,726	555,143	677,358	211,875	1,419,652	672,714
DNS Response Packets	1,857,962	549,041	753,533	202,939	1,543,813	689,911

Table 3.2: *Daily DNS Traffic Averages – A comparison between the averages of DNS packets processed during weekdays vs. weekends* [6]

students and visiting exchange students during Fall then Spring and not many classes are scheduled during Summer when students are on vacation.

January 1, 2016 - March 31, 2017						
	Spring (Jan. to Apr.)		Summer (May to Aug.)		Fall (Sept. to Dec.)	
	Monthly avg.	stdev	Monthly avg.	stdev	Monthly avg.	stdev
DNS Query Packets	42,383,404	11,047,856	35,046,415	5,680,691	50,493,527	4,818,243
DNS Response Packets	46,735,649	9,451,150	38,198,061	5,522,259	53,693,549	4,916,530

Table 3.3: *Monthly DNS Traffic Averages Grouped By Academic Semester – Spring, Summer, and Fall semesters* [8]

Figures 3.2 & 3.3 represents the counts of DNS queries and responses in graphical format. The plot corresponds with Table 3.3 which shows that Fall had the highest counts, followed by Spring and then Summer.

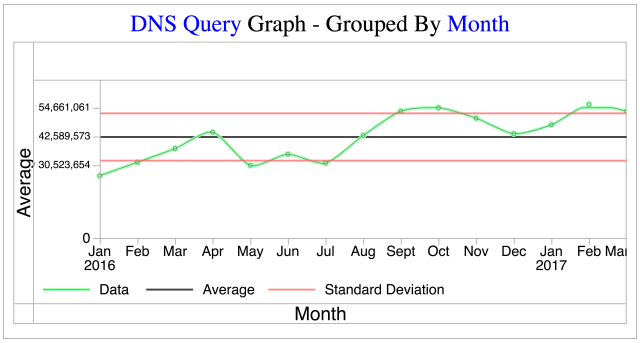


Figure 3.2: *Overall monthly distribution of all the DNS queries from January 1, 2016, to March 31, 2017* [10]

[8] Reproduced with permission from [31]

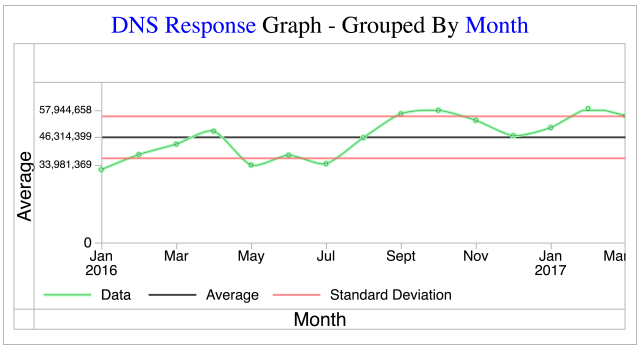


Figure 3.3: *Overall monthly distribution of all the DNS responses from January 1, 2016, to March 31, 2017* [12]

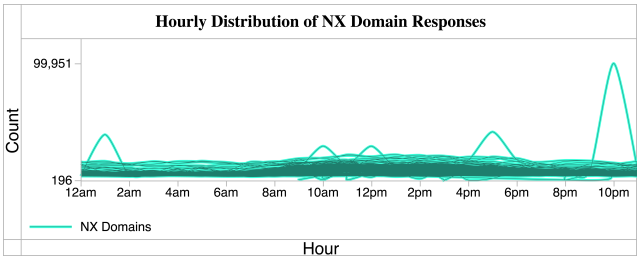


Figure 3.4: *Overall distribution of all the NXDOMAIN responses from January 1, 2016, to March 31, 2017. Note the outlier from 9:00 pm to 10:00 pm.*[14]

3.2 Notable Findings

3.2.1 WPAD queries

Web Proxy Auto-Discovery (WPAD) protocol is used by OS to automatically find and use web proxy settings. Interestingly, 1.5% of the total DNS queries were for domains with "WPAD" prefix. Because our university does not use web proxies, there are no valid internal domains starting with "WPAD". Still, more than 10 million queries were made, and if there was a valid domain, the count would have put it at bottom of top-20 most queried domains, which is higher in rank than well-known web sites like Twitter, YouTube, Amazon, Netflix.

[10]Reproduced with permission from [31]
[12]Reproduced with permission from [31]
[14]Reproduced with permission from [31]

To find the cause of such high query count, we set up three test hosts with Windows 10 Student Edition, Ubuntu 16.4 and macOS El Capitan and tried to analyze DNS queries made by them. We found that with default configuration Windows hosts were making DNS queries with "wpad" prefix. The query starts with "wpad" prefixed to the internal domain name assigned by DHCP. So for example, if the internal domain name is "dept-a.university.edu", then the first query would be to "wpad.dept-a.university.edu". If no reply is received, the first level is dropped, so it becomes "wpad.university.edu". The process continues until the domain is found or TLD is reached. As our campus does not use any web proxies, a valid reply is never received and explains the high number of such queries.

As shown by Gostev[33] and advised by US-CERT,[34] a rogue host in the network can intercept those WPAD queries and serve malicious *Proxy Auto-Configuration* (PAC) JavaScript file which the hosts will start using and send all web traffic through the rogue host, allowing Man-In-The-Middle (MITM) attack against the vulnerable host.

The eyeDNS framework helped uncover a risky default configuration which would have allowed a malicious attacker to perform a MITM attack. We have informed our university IT security team about the finding and they are working on remediation.

3.2.2 CCleaner

On October 13th, 2017, during checking the gray list domains generated by our framework we noticed a very unusual domain name `ab1145b758c30.com` in our gray list. A quick web search revealed that it was one of the lists of 11 domains used in CCleaner distribution attack.[35] CCleaner is a very popular Windows system maintenance software which had more than 2 billion downloads by November of 2016. CCleaner version 5.33 which was released on August 15, 2017, had included malicious code. The installer executable was signed by Piriform, the developing company and was hosted on their own website. The authors at Talos believe that most likely explanation is that a compromised developer account was used to insert malicious code in build process which ended up being released. Another possibility as per the authors is an insider who intentionally inserted the malicious code in the software

build process. In both cases, this was one of the first *"Supply Chain"* attacks on a popular and widely used software.

Using our eyeDNS interface we looked up remaining domains used by the malware and found `ab890e964c34.com` also exists in our data. The malware code used Domain Generation Algorithm (DGA) to calculate a domain name for each month. For example, `ab1145b758c30.com` was DGA domain for September, while `ab890e964c34.com` was for October. Our data shows that `ab1145b758c30.com` was queried only once in September and `ab890e964c34.com` was queried only once in October, which matches with the findings by the authors from Talos. Once a domain is calculated, the malicious code queries the domain.

Thanks to host level DNS log data stored and displayed by the eyeDNS framework, it became possible to identify exactly which hosts made those queries and hence most likely were infected with the malicious code. We informed our university IT Security team of our findings along with the IP address of the hosts and date-time the queries were made, using which they were able to locate the infected hosts and take necessary disinfection steps.

During a similar exercise next month we found `ab3d685a0c37.com` in our data which was the C2 domain for November. The details pointed to another host which was infected but did not communicate with the C2 server. We believe this may have in the situation where someone had reinstalled from an already downloaded infected installer.

Fortunately, we did not find any of those listed domains in December which indicates all infections were removed.

3.2.3 Other Anomalies

`wdgserv.com`

Although we used the black-list only as a guideline, we observed that `wdgserv.com` was consistently the most queried blacklisted domain in our data with 1,717 unique hosts generating 323,138 queries during the monitoring period of 15 months.

At first glance, the website `www.wdgserv.com` was displaying *Adobe Flash*-based Bible trivia game where visitors can test their knowledge and record their scores on a leader-board.

This posed a very puzzling situation for us, as it was hard to believe that a trivia game can be so popular on campus even though we had never heard about it. Moreover, the high-scores on the leader-board were very low indicating the possibility that not many people actually go to the site and play this game.

Upon further investigation, we discovered that `wdgserv.com` is a part of an advertisement and content delivery platform called *"Ask Applications"*, a subset of IAC Applications, used for distribution of interactive advertising.[36] We also discovered that `wdgserv.com` was often associated with tool-bars created by *"Mindspark"* as a part of *Ask Applications*.[37]

To determine if these tool-bars were the potential source of the large amounts of queries being observed, we set up a controlled virtual-machine environment and installed three different tool-bars created by the company: *"BringMeSports"*, *"RadioRage"*, and *"WeatherBlink"*. We observed that the tool-bars include a weather widget which submits DNS queries to `weatherblink.wdgserv.com` every time the page in the browser is refreshed or a new tab is opened. This was confirmed by installing the *"RadioRage"* toolbar. This toolbar does not include the weather widget and so no DNS queries were observed. Moreover, we encountered various articles and blog posts[38–40] regarding the removal of these tool-bars citing the tool-bars as Potentially Unwanted Programs (PUPs). These results led us to believe that one potential source for the large amounts of queries to `wdgserv.com` is tool-bars such as these present on hosts running in university network.

Through this investigation, we've shown the ability to spot suspicious activities on the eyeDNS network and measure the extent to which it is affecting the network.

Chapter 4

Limitations and Future Work

4.1 Limitations

In this section, we discuss the limitations of our framework and ways to overcome them.

4.1.1 Limited domain name logging

The Logger only log domain names up to the second level. This is based on the assumption that, if the first level domain is benign, all sub-domains under first level domain must be benign. For example, if `benign.com` is benign and `sub1.benign.com` is a sub-domain under the main domain, then the Logger assumes that `sub1.benign.com` is also benign. Similarly, when the first level domain is malicious, all sub-domains under it is assumed to be malicious as well. For example, `sub1.malicious.com` which is under `malicious.com` is malicious if `malicious.com` is malicious. Although the second scenario may be true in most cases, the first scenario may not be. For example, public hosting services such as `blogger.com`, `wix.com`, generally use second-level domains to host content for different users. So, `blogger.com` is benign, however, `malicious.blogger.com` may not be.

4.1.2 Hosts behind departmental DNS servers

When a host uses a departmental DNS server, the logger does not see the IP address of the actual host making the DNS query. Rather, to the logger, it looks like the departmental server is making the query. This causes the framework to show the departmental server as making malicious queries, where, in fact, it is a different host behind the departmental DNS server which is the culprit. As the departmental servers do not have DNS logging turned on, we are unable to find which host actually is responsible for the query. Also, the ownership and operation of the servers by departmental personnel makes it difficult to apply single logging policy which can allow the IT security team to turn DNS logging on or off as required, making it further challenging to log and analyze the DNS traffic coming through those servers.

4.1.3 Public Recursive DNS Servers

Users can configure their hosts to use public recursive DNS servers. For example, Googles `8.8.8.8`, CloudFlares `1.1.1.1`. As the DNS traffic to and from these hosts bypasses the campus DNS servers, the logger never sees the DNS traffic and hence cannot provide any analysis about the behavior of these hosts. Though systems owned by the university does not have this issue as most of them have their administrative access restricted and hence the DNS settings cannot be changed. However, currently, we do not have an estimate of on average how many personally owned systems (laptops, mobile devices, etc.) operate on campus on a typical day. We are working with our campus security team to answer this question.

4.1.4 Not actual real-time

The Logger [2.1] can be configured to send *syslog* messages as soon as it sees a specific type of packet. However, the type of packet for which it does so is limited. For example, when the Logger sees NXDomain response packet, it can send a ‘syslog message with information about the host that made the query and the domain it queried. Although, this is helpful but

does not count as actual real-time which may be required in certain cases. For example, we may want to send an alert to security team when a host queries a domain which we know is used for spreading malware. By the time the alert is sent, the host may already have resolved the domain and downloaded the malware.

4.2 Future Work

In this section, we discuss the future work that will make the framework adapt to upcoming changes and address some of the limitations that were discussed in Section-4.1

4.2.1 Anycast

Our university is in the process of switching to *"IP anycast"*[41] based DNS infrastructure. In *"IP anycast"* environment, multiple physical DNS servers with the same IP address are placed across the network. When a host makes a DNS query, it is routed to the nearest DNS servers in terms network hops. This distributed nature allows load balancing and redundancy and simplifies configuration as a single client DNS configuration works across networks. However, it also makes central and real-time logging infeasible if not impossible by our current central logger.

Fortunately, the anycast-based physical DNS servers deployed by the network team can be configured to send their DNS logs to a specified location. Our network team has set up an "ELK Stack"[42] which consists of "Elastic Search"[43], "Logstash"[44] and "Kibana"[45] and configured it to receives DNS logs from physical DNS servers all over the network. The physical DNS servers are being configured to send logs to the ELK stack. This new setup has created an opportunity for us to redesign our whole setup to work in an anycast-based environment. As more and more sub-networks switch to the new anycast-based DNS servers, the coverage of campus DNS data in the ELK stack will increase which in turn will increase the coverage of eyeDNS framework which was monitoring only 15% of university traffic. This was not possible earlier with Logger because we had only one hardware based logger

monitoring one of three main campus DNS servers. 2

Below is a road-map of our future work in this aspect that needs to be completed

1. Extract logs for a specific duration from ELK stack and generate the log files in the same format as generated by current Logger. Ideally, we would want to directly use log data from the ELK stack to produce the same graphs and tables. However, that would require us to throw away our existing code-base start from scratch.

2. Measure the delay between a DNS transaction happening and the same transaction log appearing in the ELK stack to measure minimum delay. While it is expected that there will be a delay in the log appearing in ELK stack due to both network delay of log reaching from source physical DNS servers to ELK stack and the same log being processed by the ELK stack itself (which will have to process logs from multiple physical DNS servers). The measurement will give us an idea of how close to real-time we can get and variance of such delay during a busy and non-busy time of a day.

We are currently working on Step-1.

4.2.2 Anomaly Detection using Machine Learning

Currently, the university network security team has to manually review the visual graphs and tabular data to look for anomalous behavior. We are experimenting with machine learning based algorithms to see if the process can be automated to find and send alerts to network security personnel when any anomalous behavior is detected.

4.2.3 Aesthetics Improvement

The web UI was designed to display the analyzed data in the simplest form possible. However, the design at that time did not give any focus at all on user experience as it was only being used by us. Thus navigating through the tabs, searching for data using date ranges and understanding the represented data is not very intuitive. One of our colleagues is currently

working on improving the aesthetics of the web UI to make it more easier for users to navigate and understand the displayed data.

Bibliography

[1] Root Servers. http://root-servers.org/, November 2018.

[2] IETF RFC 1039 - accessed 2/2019. Domain names - implementation and specification. https://tools.ietf.org/html/rfc1039, 1988.

[3] Maciej Korczyński, MichałKról, and Michel van Eeten. Zone poisoning: The how and where of non-secure dns dynamic updates. In *2016 Internet Measurement Conference (IMC '16)*, 2016.

[4] Oracle Dyn - accessed 4/2017. Dyn Analysis Summary Of Friday October 21 Attack. https://goo.gl/g4cTWg, 2017.

[5] Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader, and Elmar Gerhards-Padilla. A comprehensive measurement study of domain generating malware. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[6] A. Banerjee, D. Barman, M. Faloutsos, and L. N. Bhuyan. Cyber-fraud is one typo away. In *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, pages 1939–1947, April 2008. doi: 10.1109/INFOCOM.2008.258.

[7] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for dns. In *Proceedings of the 19th USENIX Conference on Security*, 2010.

[8] P. Calyam, D. Krymskiy, M. Sridharan, and P. Schopis. Active and passive measurements on campus, regional and national network backbone paths. In *14th International Conference on Computer Communications and Networks (ICCCN)*, 2005.

[9] Etienne Stalmans. A framework for dns based detection and mitigation of malware infections on a network. In *Information Systems Security Association*, 2011.

[10] Bill Horne - accessed 5/2017. Collecting, Analyzing and Responding to Enterprise Scale DNS Events. https://pdfs.semanticscholar.org/9a16/d399ac85be28b03e951e5ddc72cda3d279f0.pdf, 2017.

[11] IETF RFC 6761 - accessed 2/2019. Special-use domain names. https://tools.ietf.org/html/rfc6761, 2013.

[12] List of Top-Level Domains. https://data.iana.org/TLD/tlds-alpha-by-domain.txt, February 2019.

[13] ICANN Managed Root Server Locations. https://www.dns.icann.org/imrs/locations/, February 2019.

[14] ISC RFC 1464 - accessed 2/2019. Using the domain name system to store arbitrary string attributes. https://tools.ietf.org/html/rfc1464, 1993.

[15] Verify your site ownership. https://support.google.com/webmasters/answer/9008080, February 2019.

[16] Domain Verification. https://developers.facebook.com/docs/sharing/domain-verification, February 2019.

[17] ISC RFC 7208 - accessed 2/2019. Sender policy framework (spf) for authorizing use of domains in email, version 1. https://tools.ietf.org/html/rfc7208, 2014.

[18] ISC RFC 7489 - accessed 2/2019. Domain-based message authentication, reporting, and conformance (dmarc). https://tools.ietf.org/html/rfc7489, 2015.

[19] IETF RFC 6895 - accessed 2/2019. Domain name system (DNS) IANA considerations. https://tools.ietf.org/html/rfc6895, 2013.

[20] H. Choi, H. Lee, H. Lee, and H. Kim. Botnet detection by monitoring group activities in dns traffic. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, 2007.

[21] Nan Jiang, Jin Cao, Yu Jin, Li Erran Li, and Zhi-Li Zhang. Identifying suspicious activities through dns failure graph analysis. In *Proceedings of the The 18th IEEE International Conference on Network Protocols*, 2010.

[22] Babak Rahbarinia, Roberto Perdisci, and Manos Antonakakis. Efficient and accurate behavior-based tracking of malware-control domains in large isp networks. *ACM Trans. Priv. Secur.*, 19(2):4:1–4:31, August 2016. ISSN 2471-2566. doi: 10.1145/2960409. URL http://doi.acm.org/10.1145/2960409.

[23] Andrii Shalaginov, Katrin Franke, and Xiongwei Huang. Malware beaconing detection by mining large-scale dns logs for targeted attack identification. In *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 2016.

[24] Ricardo Villamarín-Salomón and José Carlos Brustoloni. Bayesian bot detection based on dns traffic similarity. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, 2009.

[25] Sandeep Yadav and A. L. Narasimha Reddy. Winning with dns failures: Strategies for faster botnet detection. In *SecureComm*, 2011.

[26] G. Zhao, K. Xu, L. Xu, and B. Wu. Detecting apt malware infections based on malicious dns and traffic analysis. *IEEE Access*, 3:1132–1142, 2015. ISSN 2169-3536. doi: 10.1109/ACCESS.2015.2458581.

[27] E. Stalmans and B. Irwin. A framework for dns based detection and mitigation of malware infections on a network. In *2011 Information Security for South Africa*, pages 1–8, Aug 2011. doi: 10.1109/ISSA.2011.6027531.

[28] Matthew Thomas and Aziz Mohaisen. Kindred domains: Detecting and clustering botnet domains using dns traffic. In *Proceedings of the 23rd International Conference*

on World Wide Web, WWW '14 Companion, pages 707–712, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2745-9. doi: 10.1145/2567948.2579359. URL http://doi.acm.org/10.1145/2567948.2579359.

[29] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting malicious flux service networks through passive analysis of recursive dns traces. In *2009 Annual Computer Security Applications Conference*, pages 311–320, Dec 2009. doi: 10.1109/ACSAC.2009.36.

[30] Deliang Chang, Qianli Zhang, and Xing Li. Study on os fingerprinting and nat/tethering based on dns log analysis. In *Research and Applications of Internet Measurements (RAIM)*, Internet Research Task Force and Internet Society Workshop, 2015. URL https://irtf.org/raim-2015-papers/raim-2015-paper21.pdf.

[31] C. Chowdhury, D. A. Hahn, M. R. French, E. Y. Vassermann, P. K. Manadhata, and A. G. Bardas. eyedns: Monitoring a university campus network. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7, May 2018. doi: 10.1109/ICC.2018.8422082.

[32] Apereo - accessed 9/2018. CAS Client - mod_auth_cas. https://apereo.atlassian.net/wiki/spaces/CASC/pages/103252587/mod+auth+cas, 2018.

[33] Alexander Gostev. Flame: Replication via Windows Update MITM proxy server. https://goo.gl/9FPvHf, April, 2018.

[34] US-CERT. Alert (TA16-144A) WPAD Name Collision Vulnerability. https://www.us-cert.gov/ncas/alerts/TA16-144A, April, 2017.

[35] Edmund Brumaghin, Ross Gibb, Warren Mercer, Matthew Molyett, Craig Williams. CCleanup: A Vast Number of Machines at Risk. https://blog.talosintelligence.com/2017/09/avast-distributes-malware.html, September 2017.

[36] IAC Applications - accessed 5/2017. Ask Applications. http://www.iacapps.com/brands/browser/, 2017.

[37] Pieter Arntz - accessed 5/2017. Mindspark toolbars. https://blog.malwarebytes.com/cybercrime/2014/11/mindspark-toolbars/, 2017.

[38] Alice Woods - accessed 5/2017. WeatherBlink Toolbar. How to remove? (Uninstall guide). http://www.2-spyware.com/remove-weatherblink-toolbar.html, 2017.

[39] Chona Esjay - accessed 5/2017. Remove WeatherBlink Toolbar. https://malwarefixes.com/remove-weatherblink-toolbar/, 2017.

[40] Tomas Meskauskas - accessed 5/2017. Mindspark toolbar. https://www.pcrisk.com/removal-guides/7420-mindspark-toolbar, 2017.

[41] Wikipedia - accessed 9/2018. Anycast. https://en.wikipedia.org/wiki/Anycast, 2018.

[42] elastic. Elastic Stack. https://www.elastic.co/elk-stack, February 2019.

[43] elastic. Elastic Search. https://www.elastic.co/products/elasticsearch, February 2019.

[44] elastic. Logstash. https://www.elastic.co/products/logstash, February 2019.

[45] elastic. Kibana. https://www.elastic.co/products/kibana, February 2019.

Appendix A

WinSCP script

Listing A.1: *Code snippet from WinSCP script which transfer file from Microsoft Windows to Ubuntu linux*

```
option echo off

# Force binary mode transfer
option transfer binary

# WinSCP batch mode should not ask for anything
# if it does something is not correct, so abort
option batch abort

# Establish connection
# key based authentication
open sftp://user@10.10.10.10:22 -hostkey="ssh-rsa 2048
    cf:53:4a:7f:cd:ba:e2:xx:xx:xx:xx:xx:xx:xx:xx:xx"
    -privatekey=C:\Project\user.ppk

echo Connection Established
```

```
cd /project/uploads
echo Remote Directory is
pwd

lcd D:\project
echo Local Directory is
lpwd

# Upload all text files
put -filemask="*.txt" D:\project\*.txt

# close the connection
close

# Exit WinSCP
exit
```