DNS Anomaly Detection

Hidde van der Heide, Nick Barendregt

February 6, 2011

1 Abstract

While intrusion detection systems are the basis of every security aware organization and in the past have successfully mitigated most network based threats, the signature based detection has one major drawback: The system is always one step behind the newest threats. A more abstract analysis is needed to be able to detect all network based anomalies that could indicate active malware.

Statistical analysis over a larger set of data has the advantage of detecting time based anomalies and detecting out of the ordinary activity. If this information is combined with the current real-time detection methods, a better assumption could be made on the threat of an anomaly.

We have developed a number of detection methods, both real-time and statistical analysis methods, that can detect all known DNS anomalies. These methods will be combined with a scoring mechanism that can be tuned to give the best results in different environments.

2 Introduction

Botnets have been a threat to network and system security for several years. In the last two or three years the botnet programs have become much more sophisticated. The conventional methods of detecting a virus with a local virus scanner or their spreading with Intrusion Detection Systems (IDS) will not mitigate the complete threat. New detection methods are under heavy research but there is no single widespread solution yet.

A feature that distinguishes botnets from conventional viruses is their frequent communication with a central server or other bots. The characteristics of this traffic could be used to detect botnet zombies. Most Internet communication starts with one or more Domain Name System (DNS) lookups. Command and Control (C&C) traffic used by malware is no exception to this behavior. Some types of C&C traffic such as Internet Relay Chat (IRC) is ofter blocked by internal corporate firewalls making it invisible for border IDS systems. In these cases infections could probably still be detected by analyzing DNS requests originating from the corporate network.

We will examine the feasibility of detecting malware infected systems using DNS log data or captures of DNS traffic and develop a scheme for detecting these anomalies in DNS traffic. This could include anomalies in the actual content of the request (e.g. hostnames, RR) and meta-information such as timing and re-occurrence of specific lookups.

This assignment will be done in cooperation with the company Fox-IT. Fox-IT offers a managed security monitoring service. The monitoring probes only monitor and analyze traffic. These systems are not connected to the Internet and are therefore unable to gather more information than the captured data itself. We will keep this limitation in mind while developing our methods and proof of concept.

3 Previous Work

Some research has been done in the field of DNS anomaly detection. Most of this research is focused on detecting specific kinds of botnets and malware or focuses on a single method. Since our goal is to combine the most promising methods to get the best performing DNS anomaly detector we researched a lot of different methods. This section will describe the most interesting of these methods and research into techniques needed for a better understanding of botnet operation.

A promising method for identifying infected hosts in a network is detecting their scans for new vulnerable systems. A malicious program often scans IP ranges internal and external to the network for possible targets. The scans on external ranges can be detected on the network border. Nearly all connections from the internal network to an external destination are initiated by a DNS request. A scanning worm will skip this step and initiates a large number of connections without a DNS request. Whyte et al. [16] and Hamad Binsalleeh and Amr Youssef [8] have researched this method. Since this method also requires information on all new initiated sessions, which is not available in the DNS logs, we will not research this method further.

A popular and relatively new technique used by cyber-criminals to hide their critical systems is fast-flux. The ICANN Security and Stability Advisory Committee [14] released a paper giving a clear explanation of the technique. Jose Nazario and Thorsten Holz [13] did some interesting measurements on known fast-flux domains.

Villamarn-Salomn and Brustoloni [15] focused their detection on abnormally high or temporally concentrated query rates of dynamic DNS queries. Their research was ineffective as a single detection mechanism since some legitimate domains use this technique as well for legitimate purposes. Their methods might be more successful in combination with other methods we will investigate.

The research by Choi et al. [10] created an algorithm that checks multiple botnet characteristics. The detection is based on Dynamic DNS, fixed group activity and a mechanism for detecting migrating C&C servers. They state their method works properly but the processing time might be an issue on a large scale implementation.

Kenton Born and David Gustafson [9] researched a method for detecting covert channels in DNS using character frequency analysis. This method could detect patterns that do not seem to represent a language and are therefore anomalous.

We used a list of one million most frequent used domain names from Alexa [6] to compare our results to relatively normal requests. The list is based on the data gathered from users using the Alexa.com toolbar and other sources which are not specified. The method used can not be verified but we are reasonably certain the domains are all legitimate and the list gives a good overview of frequently used domains.

4 Methods

We identified two groups of detection methods. The first group of methods analyzes the packets for anomalies in the data they contain. These detection methods can be performed as soon as the packets arrive. We will call them the online or real-time methods. The second group of methods perform statistical analysis on a large set of data. This allows us to detect anomalies in the volumes of queries or the query responses over time. These analysis methods will be performed off-line.

4.1 DNS Anomaly Detection

We researched a number of analysis methods that are relatively easy to implement and can be performed online, on a small set of data, as soon as the data is available. We focus on the data send to or from TCP or UDP port 53. The very first analysis method is to detect if we can find DNS data in the packet. Any packet without DNS data is suspicious since we found no good reason for non-DNS packets to use port 53. This will trigger an immediate alert and further checks will be skipped since there is no DNS data available.

Most LANs use internal mail servers or a mail server hosted by their ISP to send e-mails. These mail servers will deliver the e-mails on behalf of the client to the destination domain. A client PC has therefore no reason to query MX records for domains. An MX query could therefore indicate the client is infected with a mass-mailer worm. This method will also trigger an immediate alert and further checks will be skipped.

Once we identified a proper DNS query or response the data strings will be checked against a pre-configured list of keywords. This method is very similar to a domain blacklist but is not limited to a list of known malicious domains. This method will probably identify mostly human trigger anomalous DNS traffic, e.g. an network administrator could filter all requests with the word "casino" in the domain.

4.2 Blacklisting

In general a blacklist is used to deny access to certain hosts because they are known to be malicious. By combining different popular blacklists [12, 11, 4, 5, 3] we created a large blacklist to check the DNS queries against. This blacklist checking mechanism is based on the principle: If its on the list, it is malicious. We use this method to indicate a first level of malicious traffic. With the use of multiple blacklists and combining them, a broader blacklist can be generated against multiple threats. We include this method in our analyze program because it is an easy method to implement and a method not worth forgetting in our project.

Since this combined blacklist contained about 10.000 entries, which seemed to be a lot, we wondered if all these domains still existed. With a simple bash script we started our first resolving round. After 6 hours the script was finally done resolving and we had our results. In those results where a lot of weird entries, under them: 127.0.0.1, 127.0.0.2, 8.8.8.8 and 4.4.4.4 for example. Not just those strange IP addresses but also the domains with more than five assigned ip-addresses triggered our attention.

Looking closer to the domains with a lot of IP addresses we also noticed that the domain had very short Time To Live (TTL) value's for their domain names. Searching for further documentation it brought us to the term fast-flux, this will be explained in section 4.5.

4.3 Tunnel DNS Detection

DNS packets can be used to create a hidden data channel (covert channel). There are a number of ways to hide data in seemingly legitimate DNS packets. They all require a modified DNS client and server but are designed to work with legitimate, unmodified recursive DNS servers. The covert channel can be used to smuggle secret information past the firewalls designed to block the information from being spread. Since DNS data is often poorly monitored and often allowed to pass through the firewall, it is an ideal candidate for a covert channel.

The client will be configured with a domain that the modified name server is the authoritative name server for. The client encodes the upstream data in a new Lowest Level Domain (LLD) added to the configured domain and sends the query. The name server has multiple options for hiding the downstream data. It can encode the data in all the different record types with different available bandwidth, in order from high to low:

- NULL
- TXT
- SRV
- MX
- CNAME
- A

The detection of a covert channel in DNS is based on two analysis methods. The first method analyzes the packet characteristics and the second method analyzes the data. The latter will be explained in section 4.4, we will now focus on the former.

There are a few characteristics of the DNS packet that could distinguish a packet used in a covert channel. If the channel is used to transport reasonable amounts of data, the length of the LLD in the query or the size of the response record can easily be checked. The average length of the one million most used domains is ten characters, where the average query length of a DNS tunnel used to copy a random file is well over 30 characters. If this method is used in combination with timing analysis, a high volume stream with large DNS packets can be identified.

Anomalies in the record types of the response packets can also easily be identified. The NULL record is an experimental record and should not be used in production environments and as described in section 4.1, most client systems have no reason to query an MX record.

Listing 1: Tunnel DNS packet example

```
a.b.c.d  e.f.g.h  DNS  Standard  query         A
    lac0qdyn5jsibr4gbiaop5gt0ngok4cq.moscow.nexuz.net
e.f.g.h  a.b.c.d  DNS  Standard  query response  CNAME
    hgeyc2mbogaxdcljrgaxdalrqfyzc0mjrgmyc0mrx.we
```

4.4 Character Frequency

The character frequency analysis method is a very promising method for detecting generated data where a natural language would be expected. The method has successfully been used in cryptography for detecting language characteristics in a cipher text. We used the method much in the same way. Domain names are strings mostly chosen by humans and should be recognizable by humans. Therefore the domain names consist of one or more words and therefore closely follow the natural language characteristics. If the language of the users can be predicted, the characteristics of the language can be compared with the domain names the users system requests and an assumption can be made if the domain name is generated by the system or chosen by the user.

This method was initially investigated for a reliable covert channel detection but we think it should be able to detect more anomalous data. We have covered DNS tunnel detection in section 4.3. The data hidden in DNS is often normal text and therefore follows the natural language characteristics. It is however very likely the adversary would compress and encrypt the data to optimize the bandwidth and hide the smuggled data. The encryption will by definition obfuscate the natural language characteristics and this could be exploited by our detection method.

A different implementation could be the detection of anomalous domain names. Botnets C&C servers often change domain names and therefore use random strings. This could be detected but will be a challenge. The data of a single domain is very limited and would probably not contain all possible characters. If this data is stored and we can analyze a number of domains, the detection could be a lot more certain. The obvious trade off is the time it takes before the anomaly is detected.

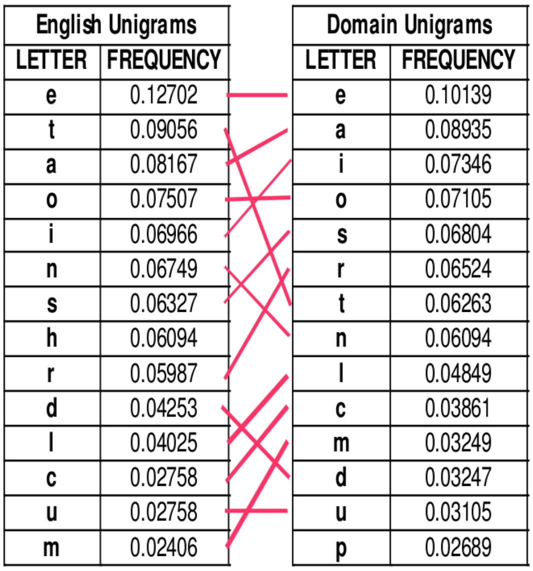
4.4.1 Top Domain N-gram analysis

The method has been researched by Kenton Born and David Gustafson [9]. We used the results and conclusions from this paper after verifying them. Character frequency analysis in languages and cryptography has been studied for centuries. We used the English frequency table published by Henry Beker and Fred Piper [7].

Kenton Born and David Gustafson compared the letter frequency table for the English language with a list of one million most frequent used domains [6]. We reproduced this test and verified the results.

It can be clearly seen in the unigram table in figure 1 that the domain letter frequencies are not equal but very similar to the English language. Most letters are on similar places in the table with the biggest anomalies being the letters "H" and "T". This can be explained by the lack of the word "the" in most domain names.

Figure 1: English Domains Unigram

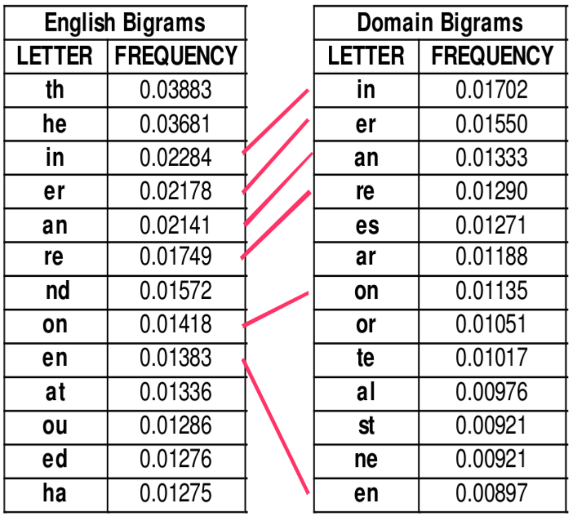


English Unigrams	
LETTER	FREQUENCY
e	0.12702
t	0.09056
a	0.08167
o	0.07507
i	0.06966
n	0.06749
s	0.06327
h	0.06094
r	0.05987
d	0.04253
l	0.04025
c	0.02758
u	0.02758
m	0.02406

Domain Unigrams	
LETTER	FREQUENCY
e	0.10139
a	0.08935
i	0.07346
o	0.07105
s	0.06804
r	0.06524
t	0.06263
n	0.06094
l	0.04849
c	0.03861
m	0.03249
d	0.03247
u	0.03105
p	0.02689

Source: Detecting DNS Tunnels Using Character Frequency Analysis [9]

Figure 2: English Domains Bigram

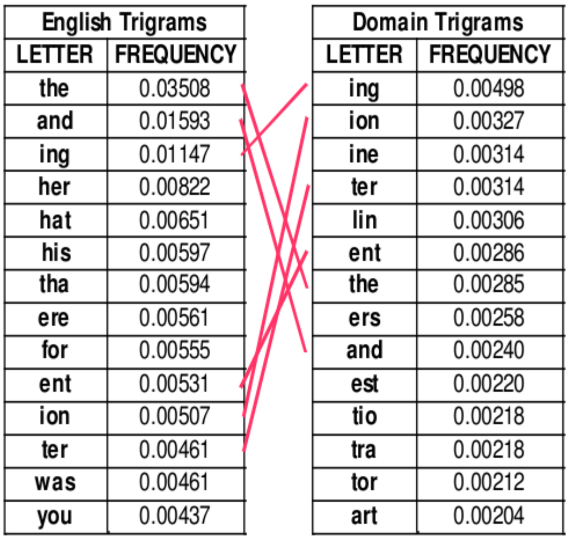


English Bigrams	
LETTER	FREQUENCY
th	0.03883
he	0.03681
in	0.02284
er	0.02178
an	0.02141
re	0.01749
nd	0.01572
on	0.01418
en	0.01383
at	0.01336
ou	0.01286
ed	0.01276
ha	0.01275

Domain Bigrams	
LETTER	FREQUENCY
in	0.01702
er	0.01550
an	0.01333
re	0.01290
es	0.01271
ar	0.01188
on	0.01135
or	0.01051
te	0.01017
al	0.00976
st	0.00921
ne	0.00921
en	0.00897

Source: Detecting DNS Tunnels Using Character Frequency Analysis [9]

Figure 3: English Domains Trigram



English Trigrams	
LETTER	FREQUENCY
the	0.03508
and	0.01593
ing	0.01147
her	0.00822
hat	0.00651
his	0.00597
tha	0.00594
ere	0.00561
for	0.00555
ent	0.00531
ion	0.00507
ter	0.00461
was	0.00461
you	0.00437

Domain Trigrams	
LETTER	FREQUENCY
ing	0.00498
ion	0.00327
ine	0.00314
ter	0.00314
lin	0.00306
ent	0.00286
the	0.00285
ers	0.00258
and	0.00240
est	0.00220
tio	0.00218
tra	0.00218
tor	0.00212
art	0.00204

Source: Detecting DNS Tunnels Using Character Frequency Analysis [9]

In figure 2 a very similar anomaly can be seen. The most frequent bigrams in the English language are ”TH” and ”HE” which form the word ”the” when combined. These bigrams are missing from the most frequent domain bigrams. This table has less similarities than the unigram table but note that the bigram possibilities are exponential to the unigrams.

The last table in figure 3 is for the trigrams. As with the previous tables the trigrams for both the domains as the English language show similarities which can be exploited to detect anomalous domains.

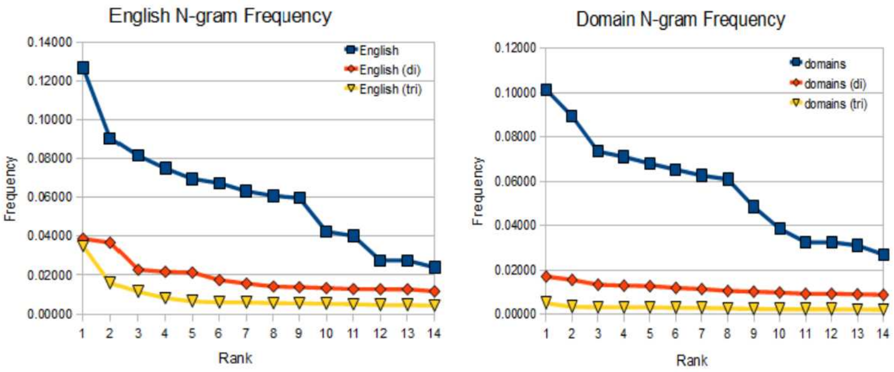
By comparing the different N-gram frequencies from the English language or top domains with a frequency table generated from the captured data, the analysis method could detect if the data is likely to be English or anomalous. By checking with different tables for different languages, the language of a domain could be predicted. However if a small set of domains is checked against a large set of different frequency tables a false positive for one of these tables is very likely.

4.4.2 Zipfian Distribution

Apart from matching frequency tables as described in section 4.4.1 a more generic detection method could be created with letter frequencies. The frequency tables are not similar for different languages but the distribution of the frequencies over the different ranks in the table are very similar. The linguist George Kingsley Zipf proposed this for the first time in 1935 though Jean-Baptiste Estoup appears to have noticed the relationship before Zipf.

Zipf’s Law [17] states that the frequency of a word is inversely proportional to its place in the frequency table. So the most frequent word will occur twice as often as the second most frequent word. This distribution is observed in many

Figure 4: English and Domain Frequency by Rank



Source: Detecting DNS Tunnels Using Character Frequency Analysis [9]

other rankings including the frequency of letters.

This means that the letter itself is not relevant for the detection method but the frequency distribution of the letters is. This is illustrated in figure 4. It can be seen that the distribution closely follows the Zipfian distribution for the English language and the distribution for the top domains also has characteristics of the Zipfian distribution.

If a similar graph would be made of an encrypted data channel the frequency distribution would by definition be almost even and the line would be a flat horizontal line. This can off-course easily be detected and works for any western language.

4.5 Fast-Flux Detection

Fast-flux is a technique used by cyber-criminals to hide critical hosts behind an ever changing set of compromised hosts. A modified DNS server is used that will return a different set of IP addresses for a given domain over time. These addresses belong to compromised hosts that will relay for instance HTTP (web) traffic to a server hosting a malicious site. This server is sometimes called the fast-flux mothership, see figure 5.

A query to a fast-flux hosting service can easily be detected. An example of a lookup for such a domain is given in listing 2. There are three things that should be noticed. The first obvious notion is the number of records. The domain resolves to ten A records which is a lot for most websites. The second notion would be the IP addresses the domain resolves to. The addresses are spread over a large number of address spaces. Then the Time To Live (TTL) of the records is abnormally low. It has been set to 300 seconds, or five minutes. This will store the results only for a short period of time in the cache of the client or its caching name server. This allows the fast-flux service provider to switch hosts very frequently, to mitigate the fact that bots are normal client PCs which are not reliable as service providers.

Listing 2: Single Flux Domain Example

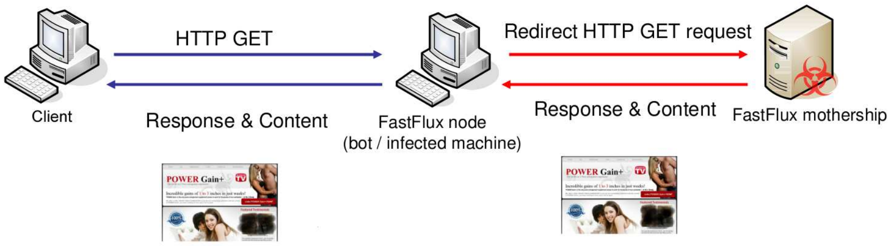
```
$ dig naughtydateingsite.net
;; ANSWER SECTION:
naughtydateingsite.net.   300   IN   A   77.127.166.235
naughtydateingsite.net.   300   IN   A   82.228.65.61
naughtydateingsite.net.   300   IN   A   84.109.81.176
naughtydateingsite.net.   300   IN   A   92.253.40.134
naughtydateingsite.net.   300   IN   A   94.54.254.3
naughtydateingsite.net.   300   IN   A   94.228.118.59
naughtydateingsite.net.   300   IN   A   114.33.131.22
naughtydateingsite.net.   300   IN   A   118.101.225.28
naughtydateingsite.net.   300   IN   A   201.167.15.123
naughtydateingsite.net.   300   IN   A   203.99.233.142
;; AUTHORITY SECTION:
naughtydateingsite.net.   172318   IN   NS   ns1.7418391.com.
naughtydateingsite.net.   172318   IN   NS   ns2.7418391.com.
naughtydateingsite.net.   172318   IN   NS   ns3.7418391.com.
naughtydateingsite.net.   172318   IN   NS   ns4.7418391.com.
naughtydateingsite.net.   172318   IN   NS   ns5.7418391.com.
naughtydateingsite.net.   172318   IN   NS   ns6.7418391.com.
;; ADDITIONAL SECTION:
ns1.7418391.com.   85917   IN   A   173.212.75.160
ns2.7418391.com.   85917   IN   A   79.119.188.9
ns3.7418391.com.   85917   IN   A   88.87.251.45
ns4.7418391.com.   85917   IN   A   82.228.65.61
ns5.7418391.com.   85917   IN   A   79.117.122.25
ns6.7418391.com.   85917   IN   A   186.114.80.139
```

There is a second fast-flux technique which is increasing in popularity, called double flux. A double flux service provider will host the domains name servers also on the botnet, mitigating the weakest link of a single flux domain. A single flux domain can be shut down by shutting down its name servers. A double flux service provider will regulary update the records at the domain registrar to reflect the changes in its botnet.

4.6 Time Based Analysis

One of the more abstract methods is the time based analysis method. This method is designed to detect anomalies in the timing of queries. By collecting the queried domains, the time they occured and the host that initiated the

Figure 5: Fast-Flux Example



Source: Abuse.ch presentation

query a number of anomalies can be detected. The most interesting anomalies we examined could all indicate botnet activety.

Most botnet programs are designed to periodically query a central server for commands to execute. The server could instruct its bots to start scanning for vulnerable systems, start sending spam e-mails or download a malicious program. Botnet herders will program the domain name of the server in the botnet client software, so the bots will be able to locate the server. To be able to connect to the server, a bot has to query the domain name and resolve its IP address.

If a group of hosts in a network have been compromised by the same botnet, an analysis method could be designed to detect similar queries from the same set of hosts within a timeperiod. A clear difference should be seen if these hosts are compared with hosts that have not been comprimised.

A different approach would be to analyse the queries over time. A host querying the same domain on a regular interval could indicate a bot querying the server for new commands. This method would not work very well if the host has a local DNS cache that caches the query for a significant period of time.

In some environments, e.g. server environments, hosts are left powered on outside office hours. If these hosts are infected with a botnet program the software would not stop querying the server outside office hours. This could easily be detected. As with the previous approach this will not work efficiently if a query is cached locally.

4.7 Scoring Mechanism

We have examined a lot of different methods which are designed to detect all known exploitable DNS anomalies. To create a system that detects all these anomalies, the different methods have to be combined. Some methods however will generate more false positives or false negatives than an other method. If all these methods would have the same alerting level, the system would be unusable. A scoring mechanism would mitigate this problem by assigning the less certain results a lower score. The system could then be tuned to work as efficiently as possible in different environments.

5 System Design

In this section we will describe what choices we have made, how our analysis program is set up and we give a general overview of how it works.

5.1 Choices

In our original design of our analysis program we made some choices based on our knowledge and based on the use within a small organization. In this first concept we did not fully thought about how to upscale our solution to an environment where the amount of DNS query's would be extremely higher. In our analysis program we decided to test everything in separate methods to focus our programming on every method separately.

Due to the limited time we decided we would focus on pre-generated network traffic in the form of pcap files. Reading pcap files instead of live traffic was an easier way to test our methods and scanning techniques than doing it on live network traffic. If we are able to proof our methods on pre-generated network traffic then it will also work on live network traffic.

5.2 Code Design

The analysis program has an highly modular design. This was done mainly to test our methods separately from each other. In our analysis program, analysis on different methods was done from the pre-generated pcap files, calling different methods read from command line parameters.

In the final version of the proof of concept, every method should be combined with a scoring algorithm to create the alerting value of a DNS query. More about the proof of concept and the scoring mechanism will be discussed in section 8.

5.3 Database Design

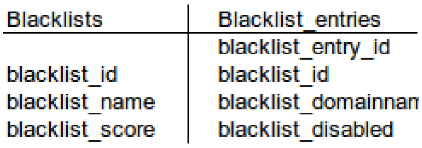
Because we choose to work with a small database to improve performance and readability of the captured data, the design was important to make it efficient as possible. To limit the size of the database on the blacklist lookup part, we decided that adding a new blacklist to the database had to be optimized. Since some blacklists contain the same domains, blindly adding a blacklist to the database was not really smart. For that problem we introduced a lookup in the database for each new entry. It will check if an entry exists from an previously imported blacklist. If the domain name already exists then the entry with the highest value would have his domain-name activated and the domain-name with a lower value would have its entry disabled.

These and similar tricks are used to shorten query time of the database and therefore resources will be less stressed. This method of checking every domain before entering into a database greatly decreases its performance when adding blacklists to the database, but will greatly increase performance while looking up data. Because the value lookup takes roughly only half of the time of a lookup on an entire database.

The database consists out of multiple tables, each with links to each-other. This has been done to separate the results or entries from the general part. Multiple blacklists can be added using this system and they only have to have one 'danger value' assigned to an entire blacklist. This value can then be changed on one record and not on every entry of the blacklist. The same model is used for the results of the analysis and reporting parts. Every instance starts with generating an ID and the start time for that round, with that unique ID results and reports will be added to a different table for the entries.

A small example of how a database design was implemented in our analysis program is show in figure 6.

Figure 6: Database Design example



Blacklists	Blacklist_entries
	blacklist_entry_id
blacklist_id	blacklist_id
blacklist_name	blacklist_domainnan
blacklist_score	blacklist_disabled

Design example for the blacklist part

6 Results

This section contains the most interesting results from the detection methods we developed and the other research we have done to support our methods and hypothesis.

6.1 Blacklisting

We implemented a blacklist checking method based on our own created blacklist. The program allows the usage of multiple blacklists and the use of different scores for each blacklist also determine what score you get back a domain. When a domain name is queried the number of entries in the blacklist table should be taken into account. In our analysis program we used a simple sqlite database setup for testing purposes. This seemed sufficient enough for our research. As the number of entries in the blacklist table grows the execution time of a query also grows. Even with the use some optimization techniques, looking up if a domain name is listed in the blacklist still takes a relatively long time if a domain name is not blacklisted. Keeping this list up to date and as short as possible is therefore a real performance challenge if the probe is under heavy DNS traffic. With a blacklist database of 10.000 entries the average throughput is 293 queries per second.

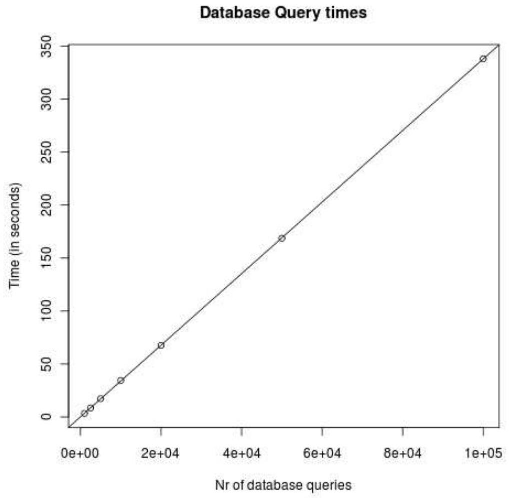
Database Query times

Queries	Time (s)	Queries a Sec.
1000	3.438	290.8667
2500	8.536	292.8772
5000	17.250	289.8550
10000	34.475	290.0652
20000	67.524	296.1909
50000	168.675	296.4280
100000	338.000	295.8579
Total average		293.1631

6.2 Fast-flux Detection

In our search to detect Fast-flux domains we already had some promising results in the first few measurements. We collected already more then 20 unique IP

Figure 7: Database Query time

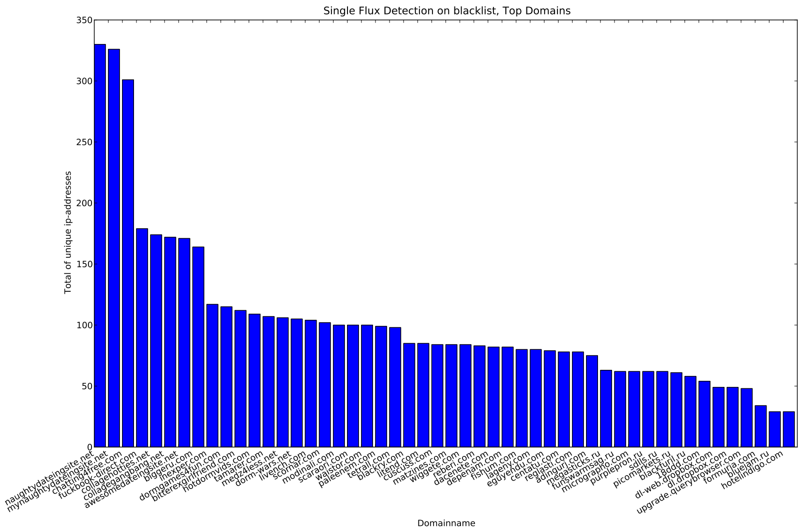


The time in seconds it takes to lookup 1.000, 2.500, 5.000, 10.000, 20.000, 50.000 and 100.000 not blacklisted domain names, with 10.000 blacklist entries in the database table.

addresses for some hosts with five test runs. After 7 days of measurements approximately every half hour for the first 2 days, every hour for the next 2 days and every 4 hours for the next 3 days, we could clearly see that some domains changed IP addresses a lot. After further investigation to these domains, they mostly contained dating related web pages. Results are plotted in figure 8.

After reading more about fast-flux we payed some attention to the double flux detection. In our search for detecting fast-flux domains we altered our code to also look for the changing name servers of listed domains. We also took the top 10.000 most visited domains from our top one million most visited domains, to test if those domains used techniques similar to fast-flux.

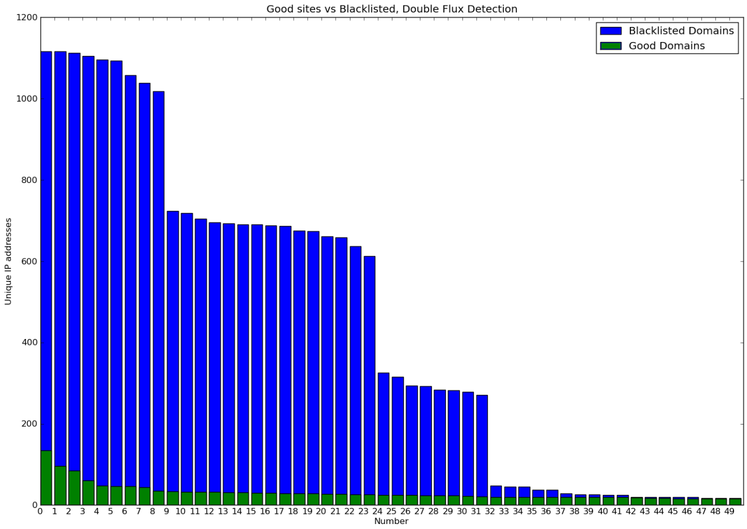
Figure 8: Single flux detection rates



Every domain is listed with the number of unique ip-addresses found for that domain.

As clearly can be seen in figure 9, the top blacklisted domains actually change ip-addresses a lot. The former top of the single flux detection method are now listed number 24 till 32 instead of number 1 till 6. With the top 8 domains in the double flux detection scoring relatively low with the single flux detection method, there is clearly a proof this detection method works quite well for detecting (double) flux domains. Noticeable is that the top 10.000 not blacklisted (good) domains have lower detection rates then the blacklisted domains. This is mainly because the good domains use higher TTL values and they normally do not have the need for changing addresses a lot.

Figure 9: Double flux detection rates



The top 50 is listed with the number of unique ip-addresses found for that domain.

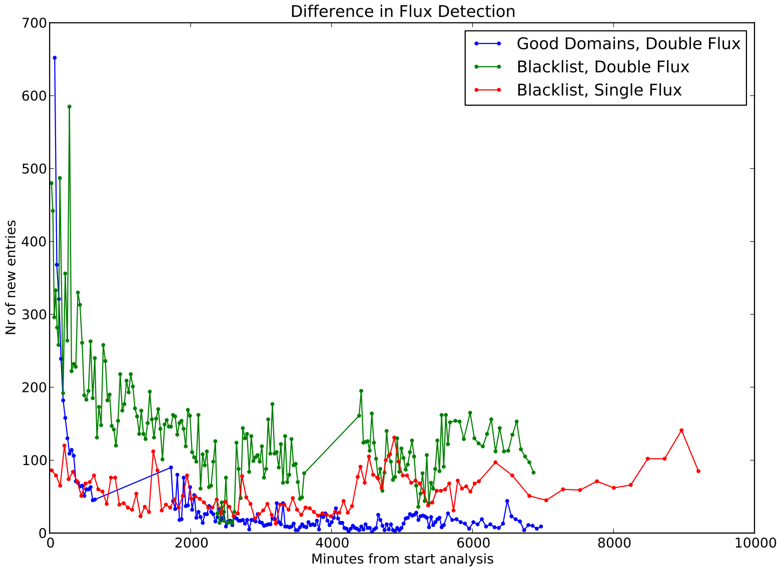
As a comparison between the two detection methods and the two lists, figure 10 gives an overview of the amount of new IP addresses collected during each run. Noticeable is that the resolving of the good domains drops to very low values within a couple of runs and then remains low. The first part has to do with the first run of the script, a lot of domains received errors and after those runs eventually the curve drops to almost no changing hosts. How ever in the characteristics of the double flux detection on the blacklisted domains, it is clearly visible that the amount of new ip-addresses collected remains higher then the rest, with some ups and downs due to expiring TTL's. In the single flux detection line there is a small change noticeable after the 4.000 minute mark. After that mark we added 20 known fast-flux domains to our blacklist and therefore the discovery of new ip-addresses was slightly higher per run.

6.3 Character Frequency Analysis

Since this method has been chosen to detect suspicious patterns in DNS to detect DNS tunnels, we focused our research on this goal. Our initial research was done on the data generated with the Iodine [1] software. We compared our results with data generated with other DNS tunnel software, but no major differences could be identified. The data was captured with tcpdump, read with the Scapy [2] library and analyzed with the NLTK toolkit.

To get a baseline for the character frequency distribution we first verified the results from the research done by Kenton Born and David Gustafson [9]. The results can be found in figure 1, 2 and 3. As described in section 4.4 the

Figure 10: Flux detection rates



Measurements over time, listed are the total new ip-addresses per run.

results are promising.

We applied the same method to compare the domain strings generated by the DNS tunnel software with the top domains list. We hypothesized that if the data is truly random, the frequency should be distributed evenly. This is however not completely the case as can be seen in figure 13. The software seems to have a preference for certain characters. This behavior can partly be explained by the fact that the software sends a frequent heartbeat with a similar string every few seconds. Some of the characters also seem to have a special meaning and are used for signaling. This has not been researched in depth.

We focused the research on data encoded with base 32 or base 128. The results for base 128 encoded data show a far superior bandwidth as expected, but characters not present at a normal keyboard show high in the frequency table (figure 12). This is a clear indication of an anomaly. The anomalies for base 32 encoded data (figure 11) are harder to detect. If you would compare the frequency distribution table with the table for the most frequent domains it is clear they are very different. A system could be made to easily detect this difference. A more language independent approach would be harder since the distribution of frequencies is not evenly as predicted. More research is needed on real live data to confirm these results.

Figure 11: Base 32 frequency distribution

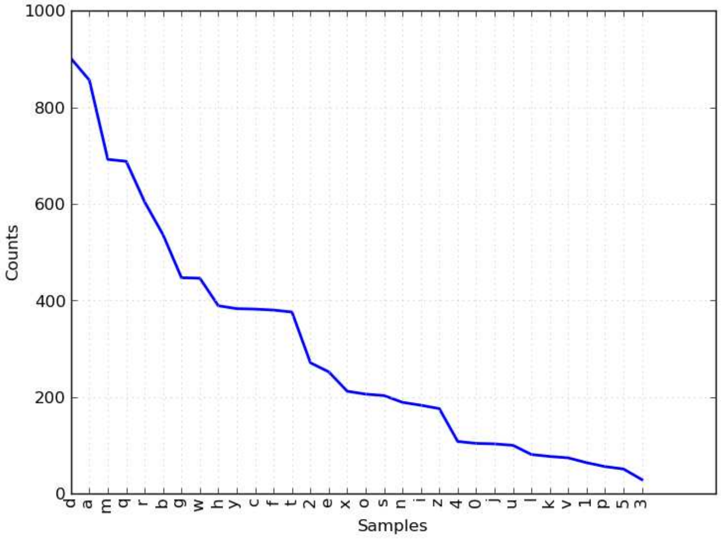


Figure 12: Base 128 frequency distribution

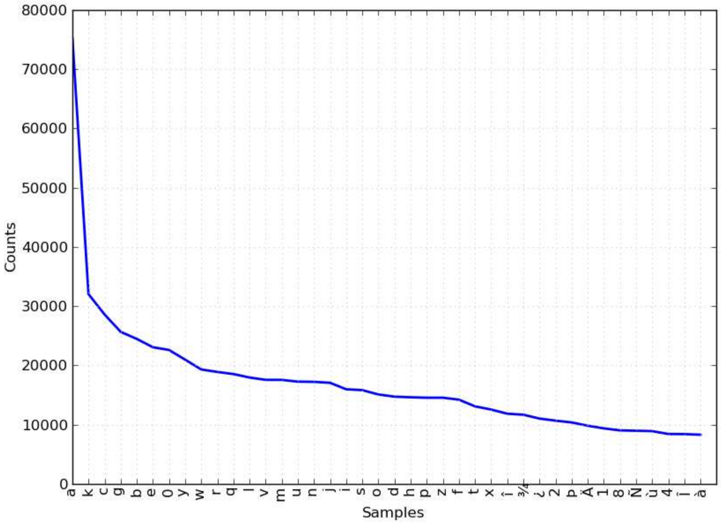
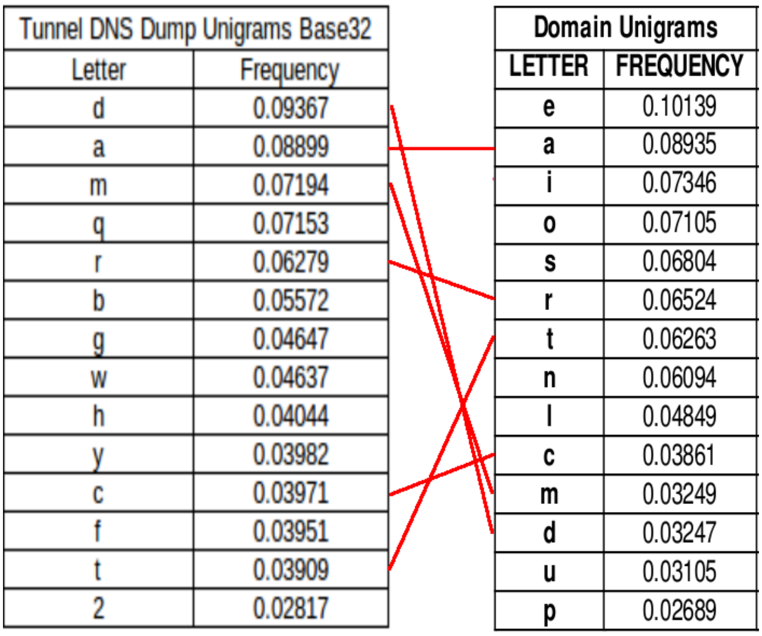


Figure 13: Base 32 comparison to top domains



Tunnel DNS Dump Unigrams Base32	
Letter	Frequency
d	0.09367
a	0.08899
m	0.07194
q	0.07153
r	0.06279
b	0.05572
g	0.04647
w	0.04637
h	0.04044
y	0.03982
c	0.03971
f	0.03951
t	0.03909
2	0.02817

Domain Unigrams	
LETTER	FREQUENCY
e	0.10139
a	0.08935
i	0.07346
o	0.07105
s	0.06804
r	0.06524
t	0.06263
n	0.06094
l	0.04849
c	0.03861
m	0.03249
d	0.03247
u	0.03105
p	0.02689

7 Conclusions

During the course of the project it became increasingly more clear that the statistical analysis methods would get the best detection results but that handling the data needed to do this analysis would be to big of a challenge to solve in the time we had. We also thought of more methods than we could implement and researching the methods took a bigger chunk of the available time than anticipated. We have therefore a lot of room for future research as described in section 8.

The first few simple methods implemented in the POC can all be done real-time, but as we started investigation the implementation of the more difficult methods it became clear doing these could not be done in real-time. The fast-flux detection method, the timing analysis method and the letter frequency method are more accurate on large sets of data and a smart decision could therefore be made on when would be the optimal time to analyze a subset of the available data.

We believe that when the methods described in this paper are combined with a scoring mechanism and the scoring mechanism is tuned with real life tests, a system is realized that could detect all known anomalies. We have covered most, if not all possible exploitable DNS vectors and a proper detection method has been proposed for all.

8 Future Work

8.1 Proof of concept

During our research project we did not had the time to create a fully working proof of concept. This proof of concept was not mandatory for our project, but it was very welcome if we succeeded in creating a working proof of concept. In this paragraph we will explain the parts we did not cover in our research, why we did not succeed and what can be done to make it a success.

8.2 Scoring Mechanism

The main idea of combining different methods to one big program is to have a solid answer to your question, in our case is this domain malicious or not. Since there is no single method that could possibly detect all known DNS vectors, different methods have to be combined.

This score is very important for an query to be marked malicious, because different methods are used and every method could produce a value according to that method. Because tuning these methods is a research project in itself, we did not implement it in full in our analysis program. The work to be done is determine the best score for every method or a combination of methods together. If the generated score then reaches a threshold an alert must be given.

8.3 Timing analysis

For a proper timing analysis a lot of data is needed over a longer period of time. Based on our previous mentioned methods in 4.6 there are some ways according to us, to correctly implement a good checking mechanism for time based analysis. Also our methods require traffic over a longer period of time. This would create a huge amount of data to be checked which is a challenging project in itself.

8.4 Character Frequency Analysis

Future work for the character frequency analysis would be to further examine the most common tools for DNS tunneling and what kind DNS traffic they generate. Over a longer period of time DNS queries should be monitored and used for a learning dataset for the character characteristics, both from the DNS tunnels as for normal DNS traffic. Eventually this character analysis should be fully implemented and be able to generate a score based on a DNS query.

8.5 Live Data

In our analysis program we only worked with pre-generated network traffic. For the real proof of concept it should also work on live data streams. As mentioned in section 5.1 we had our reasons for that. However we created the program to be able to use live data. Every method takes an input and generates an output based on the input. Together with the creation of a full working proof of concept this is a key feature for future work.

9 Bibliography

References

[1] http://code.kryo.se/iodine/.

[2] http://www.secdev.org/projects/scapy/.

[3] abuse.ch. Amada tracker. http://amada.abuse.ch, jan 2011.

[4] abuse.ch. Spyeye tracker. https://spyeyetracker.abuse.ch, jan 2011.

[5] abuse.ch. Zeus tracker. https://zeustracker.abuse.ch, jan 2011.

[6] Alexa.com. Alexa top 1,000,000 global sites. http://www.alexa.com/topsites, jan 2010.

[7] Henry Beker and Fred Piper. The protection of communications. *Wiley-Interscience*, page 397, 1982.

[8] Hamad Binsalleeh and Amr Youssef. An implementation for a worm detection and mitigation system. 2008.

[9] Kenton Born and David Gustafson. Detecting dns tunnels using character frequency analysis. *CoRR*, abs/1004.4358, 2010.

[10] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet detection by monitoring group activities in dns traffic. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, pages 715–720, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Andr D. Corra. Malwarepatrol. http://www.malware.com.br/lists.shtml, jan 2011.

[12] dglosser. Dns-bh malware domain blocklist. http://www.malwaredomains.com/, jan 2011.

[13] Jose Nazario and Thorsten Holz. As the net churns: Fast-flux botnet observations. *International Conference on In Malicious and Unwanted Software*, (3):24–31, 2008.

[14] Security and Stability Advisory Committee (SSAC). Ssac advisory on fast flux hosting and dns. 2008.

[15] Ricardo Villamarn-Salomn and J.C. Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. *Consumer Communications and Networking Conference*, (5):476–481, 2008.

[16] David Whyte, Evangelos Kranakis, and Paul C. van Oorschot. Dns-based detection of scanning worms in an enterprise network. 2005.

[17] George K. Zipf. Human behaviour and the principle of least effort: an introduction to human ecology. 1949.