Global Information Assurance Certification Paper

Detecting DNS Tunneling

GIAC (GCIA) Gold Certification

Author: Greg Farnham
Advisor: Antonios Atlasis

Accepted: February 25th, 2013

Abstract

DNS is a foundational protocol which enables applications such as web browsers to function based on domain names. DNS is not intended for a command channel or general purpose tunneling. However, several utilities have been developed to enable tunneling over DNS. Because it is not intended for general data transfer, DNS often has less attention in terms of security monitoring than other protocols such as web traffic. If DNS tunneling goes undetected, it represents a significant risk to an organization. This paper reviews DNS tunneling utilities and discusses practical techniques for detecting DNS tunneling. Two categories of detection considered are payload analysis and traffic analysis. The payload detection techniques have been used to detect successfully specific DNS tunneling utilities. The traffic analysis based technique can be used to universally detect DNS tunneling. With these detection techniques implemented organizations can reduce the risk associated with DNS tunneling.

1. Introduction

Web browsing and email use the important protocol, the Domain Name System (DNS), which allows applications to function using names, such as example.com, instead of hard-to-remember IP addresses. Because DNS is not intended for data transfer, people can overlook it as a threat for malicious communications or for data exfiltration. According to Rasmussen, for DNS threats, 'most enterprises are wide-open to real attacks via this little-known vector '(Rasmussen, 2012). Many organizations have little or no monitoring for DNS. Instead, they focus resources on web or email traffic where attacks often take place.

There are a number of tools available for tunneling over DNS which are discussed in this paper. A common motivation for these tools is to get free Wi-Fi access for sites with a captive portal for http, but free flowing DNS. These tools can also be used for more malicious activities. A DNS tunnel can be used for as a full remote control channel for a compromised internal host. Capabilities include Operating System (OS) commands, file transfers or even a full IP tunnel. For example, data exfiltration via DNS tunneling is a method incorporated in to the squeeza penetration testing tool (Haroon, 2007). These tools can also be used as a covert channel for malware. For example, Feederbot (Dietrich, 2011) and Moto (Mullaney, 2011) are known to use DNS as a communication method.

DNS tunneling poses a significant threat and there are methods to detect it. DNS tunnels can be detected by analyzing a single DNS payload or by traffic analysis such as analyzing count and frequency of requests. Payload analysis is used to detect malicious activity based on a single request. Attributes of a request such as domain length, number of bytes and content can be used to create detection rules. Detecting uncommon record types such as TXT can be used as well. The other useful method is traffic analysis. Traffic analysis is used to detect malicious activity based on multiple requests or overall traffic. Attributes that can be used for traffic analysis include volume of DNS traffic, number of hostnames per domain, geographic location and domain history.

In this paper, several DNS tunneling tools are presented and several detection techniques are discussed. To begin, an overview of DNS is provided.

2. DNS Overview

Domain Name System (DNS) is a critical protocol and service used on the internet. The most common use of DNS is to map domain names to IP addresses. Users can enter a domain name (e.g. example.com) in their web browser. DNS is used to perform a forward lookup to find one or more IP addresses for that domain name. This is known as an 'A' record. The User's network stack can than send http traffic to the destination IP address. DNS is constantly being enhanced to provide new capabilities. Although there are earlier RFCs, the core DNS functionality is defined in RFCs 1034 and 1035 (Kozierok, 2005). There are over 20 other RFCs describing additional DNS functionality.

DNS has over 30 record types with many of the common ones being critical to core internet services. As mentioned earlier, the 'A' record type maps a domain name to an ipv4 address. The 'AAAA' record is used to map a domain to an ipv6 address. The 'CNAME' record type is used to map a domain name to the canonical name. The 'MX' record type is used to define mail servers for a domain. The 'NS' record type is used to define authoritative name servers for a domain. The 'PTR' or pointer record is commonly used to map an IP address to its domain name. This is commonly referred to as a reverse lookup. The 'TXT' record type is used to return text data. This record type has been leveraged for specific purposes such as Sender Policy Framework (SPF) for anti-spam (Wong, 2006).

DNS uses both UDP server port 53 and TCP server port 53 for communications. Typically UDP is used, but TCP will be used for zone transfers or with payloads over 512 bytes. There is also the 'Extension Mechanisms for DNS (EDNS) (Vixie, 1999). If EDNS is supported by both hosts in a DNS communication, then UDP payloads greater than 512 bytes can be used. EDNS is a feature that can be leveraged to improve bandwidth for DNS tunneling.

DNS is a hierarchical system; each level in the hierarchy can be provided by another server with different ownership. For the internet, there are 13 root dns servers labeled A thru M. These are represented by many more than 13 physical servers. The hierarchical nature of DNS can be explained using an example. Take an example request

for the IP address of a domain named my.test.example.com. A new request will first go to the root servers to find which server controls the .com top level domain. The .com server will provide the server that controls example.com domain. Next, the example.com DNS server will provide the server that controls the test.example.com domain. Finally, the test.example.com DNS server will provide the IP address for my.test.example.com.

With this hierarchical system a given domain owner can define the authoritative servers for their domain. This means that they are in control of the ultimate destination host for DNS queries for their domain. In a typical enterprise, endpoints do not make DNS requests directly to the internet. They have internal DNS servers that provide DNS services to an endpoint. However, given that DNS will forward their requests until the authoritative name server is contacted, an attacker with access on an internal endpoint can leverage the enterprise's DNS infrastructure for DNS tunneling to a domain that they control.

DNS performs caching. When DNS answers are provided a time to live (TTL) is included. The receiving intermediate server can use that value for the amount of time to cache the result. Then if an identical request comes in, the cached result can be provided instead of performing another lookup.

3. DNS Tunneling

DNS is a service that is used on every system with general purpose use of the internet. It is therefore a convenient target for misuse. The misuse under consideration here is tunneling. With DNS tunneling, another protocol can be tunneled through DNS. A DNS tunnel can be used for 'command and control', data exfiltration or tunneling of any internet protocol (IP) traffic. In a 2012 presentation at the RSA conference, Ed Skoudis identified that DNS based Command and Control of malware as one of the six most dangerous new attacks. Ed shared that 'Attackers have recently used this technique in cases involving the theft of millions of accounts' (Skoudis, 2012). It has been shown that DNS tunneling can achieve bandwidth of 110 KB/s (Kilobytes per second) with latency of 150 ms (Van Leijenhorst, 2008).

3.1. Tunnel Basics

Many of the tools for creating DNS tunnels were created with the intent of bypassing captive portals for paid Wi-Fi service. If one of these systems allows all DNS traffic out, a DNS tunnel can be set up to tunnel IP traffic without paying for service. Some of the DNS tunneling utilities will create a tun or a tap interface locally on the endpoint system. There will also be a tun or a tap device on the 'DNS' server hosting the DNS tunneling tool. This will allow the user to tunnel IP traffic to the internet. This technique is similar to how VPN software works such as OpenVPN. There are even commercial service providers that provide the server side tunnel as a service. These services can be marketed as 'VPN over DNS'. The user installs the client software and connects to the service provider's 'DNS' server running the server side tunneling software. These services have client software for various Operating Systems including Android. At least in some cases they are leveraging existing DNS tunneling software such as Iodine.

3.2. Tunnel Components

The first known discussion of DNS tunneling was from Oskar Pearson on the Bugtraq mailing list in April of 1998 (Pearson, 1998). Since that time a number of DNS tunneling utilities have been developed. All of the utilities use similar core techniques but have variation on encoding and other implementation details. The core techniques used by all DNS tunneling utilities include a controlled domain or subdomain, a server side component, a client side component and data encoded in DNS payloads. The controlled domain is used to define the authoritative name server for that domain or subdomain. For discussion purposes, t.example.com will be used. The server side component will be referred to as a DNS tunnel server. The DNS tunnel server will be the authoritative name server for the controlled domain. The DNS tunnel server will typically be an internet accessible server controlled by the tunnel user. The client side component hosts the other end of the tunnel. This could be an endpoint in a security controlled enterprise environment. The tunnel could be used to communicate past the security controls and allow communication between the controlled endpoint and an arbitrary host on the internet. The client side component initiates a DNS request for which the DNS tunneling server is the authoritative name server.

3.3. Encoding and Techniques

The last core technique is to encode data in DNS payloads. This is an area where the specifics of each utility vary widely. From a high level simplified point of view, the client wants to send data to server. It will encode that data in the DNS payload. For example the client could send an 'A' record request where the data is encoded the in host name: MRZGS3TLEBWW64TFEBXXMYLMORUW4ZI.t.example.com. The server could respond with an answer as a CNAME response: NVWW2IDPOZQWY5DJNZSQ.t.example.com. In this way any data can be encoded and sent to the server. The server can also respond with any data. If there is a need for the server to initiate a communication, it cannot be done directly. Clients do not have a service listening for DNS requests and are typically behind a firewall. Server initiated communication can however be accomplished by having the client regularly poll the server. Then, if the server has data for the client it can send it as a response to the polling requests.

The implementation details are where the various DNS tunneling utilities differ. DNS utilities vary in implementation language with tools being implemented with C, Java, Perl and Python to name a few. Some utilities use a tun or tap to create a local interface and IP address for the tunnel on the hosts. Others just tunnel the binary data which can be used similar to netcat to issue Operating System commands and transfer files.

The encoding method including DNS record type is an area where tools have been implemented differently. Some utilities use common record types such as 'A' records. Others use experimental types such as 'Null' records and EDNS to improve performance (Revelli, 2009).

3.3.1. Base32 Encoding

Base32 or 5-bit encoding is commonly used for requests from the client. While DNS names can have upper case and lower case, the case is to be ignored which leaves 26 letters. Additionally, numbers and the '-' character are allowed. This provides a total of 37 unique characters. Therefore, we can take data 5 bits at a time which gives us 32 possible values. These 32 values can fit within our 37 available characters. We can then

build a string of nested subdomains out of the encoded data. DNS will allow up to 255 characters total with each subdomain (aka label) being 63 characters or less.

3.3.2. Base64 Encoding

Base64 or 6-bit encoding can be used for 'TXT' record responses. A 'TXT' record can have upper and lower case which provides 52 characters. The numbers add another 10. If we add two additional characters such as '-' and '+', we then have 64 unique values which can be used for base 64 encoding. Similar to the Base32 encoded request, the response can be encoded 6 bits at a time using a 'TXT' response and sent back to the client.

3.3.3. Binary (8 bit) Encoding

Binary 8-bit encoding can be used. The authors of Heyoka found that although it doesn't work with every DNS server, they could successfully use 8 bits per character for encoding which supports greater bandwidth through the tunnel (Revelli, 2009). Additionally, Iodine uses Null type records for responses to provide 8 bit encoding.

3.3.4. NetBIOS Encoding

NetBIOS encoding is another method of encoding data that has been used. For NetBIOS encoding, each byte is split in to 4 bit nibbles. Decimal 65 is added to each nibble. Each byte then is encoded in to two characters in a DNS label. This method is only used by DNScat-B

3.3.5. Hex Encoding

Hex encoding is another method of encoding. For hex encoding, the two character hex values are used to represent each byte. This method is only used by DNScat-B.

3.3.6. Techniques

The DNS tunneling utilities can make use of different DNS record types and encoding methods. In some cases such as iodine, the utility will auto detect the best possible encoding.

There are a couple of other implementation techniques that deserve mentioning. DNS tunneling utilities can make use of EDNS which allows them to use payloads greater than 512 bytes and thereby improve performance. One DNS tunneling utility, Heyoka will spoof the source IP addresses for requests to the server (upstream data) to lower the visibility of the client.

4. Known DNS Tunneling Utilities

There are a number of different utilities for DNS tunneling. They are summarized here.

4.1. DeNiSe

DeNiSe is a proof of concept for tunneling TCP over DNS in Python. The github page for DeNiSe has six python scripting dating between 2002 and 2006 (mdornseif, 2002).

4.2. dns2tcp

dns2tcp was written by Olivier Dembour and Nicolas Collignon. It is written in C and runs on Linux. The client can run on Windows. It supports KEY and TXT request types (Dembour, 2008).

4.3. DNScapy

DNScapy was developed by Pierre Bienaime. It uses Scapy for packet generation. DNScapy supports SSH tunneling over DNS including a Socks proxy. It can be configured to use CNAME or TXT records or both randomly (Bienaime, 2011)..

4.4. DNScat (DNScat-P)

DNScat (DNScat-P) was originally released in 2004 and the most recent version was released in 2005. It was written by Tadeusz Pietraszek. DNScat is presented as a 'swiss-army knife' tool with many uses involving bi-directional communication through DNS. DNScat is Java based and runs on Unix like systems. DNScat supports A record and CNAME record requests (Pietraszek, 2004). Since there are two utilities named DNScat, this one will be referred to as DNScat-P in this paper to distinguish it from the other one.

4.5. DNScat (DNScat-B)

DNScat (DNScat-B) was written by Ron Bowes. The earliest known public release was in 2010. It runs on Linux, Mac OS X and Windows. DNScat will encode requests in either NetBIOS encoding or hex encoding. DNScat can make use of A, AAAA, CNAME, NS, TXT and MX records. It provides a datagram and a stream mode. There is also a DNScat-B based Metasploit payload (Bowes, 2010).

4.6. Heyoka

Heyoka is a Proof of Concept which creates a bi-directional tunnel for data exfiltration. This tool is written in C and has been tested on Windows. Heyoka was developed by Alberto Revelli and Nico Leidecker. It uses binary data instead of 32 or 64 bit encoded data to increase bandwidth. It also uses EDNS to allow DNS messages greater than 512 bytes. Heyoka also uses source spoofing to make it appear that the requests are spread out over multiple IP addresses (Revelli, 2009).

4.7. iodine

iodine is a DNS tunneling program first released in 2006 with updates as recently as 2010. It was developed by Bjorn Andersson and Erik Ekman. Iodine is written in C and it runs on Linux, Mac OS X, Windows and others. Iodine has been ported to Android. It uses a tun or tap interface on the endpoint (Andersson, 2010).

4.8. NSTX

NSTX (Nameserver Transfer Protocol) From Florian Heinz and Julien Oster was released in 2000. It runs only on Linux. NSTX makes it possible to create IP tunnels using DNS (NSTX, 2002). It tunnels the traffic using either a tun or tap interface on the endpoints.

4.9. OzymanDNS

OzymanDNS is written in Perl by Dan Kaminsky in 2004. It is used to setup an SSH tunnel over DNS or for file transfer. Requests are base32 encoded and responses are base64 encoded TXT records.

4.10. psudp

psudp was developed by Kenton Born. It injects data into existing DNS requests by modifying the IP/UDP lengths. It requires all hosts participating in the covert network to send their DNS requests to a Broker service which can hold messages for a specific host until a DNS request comes from that host. The message can then be sent in the response (Born, 2010a).

4.11. squeeza

Squeeza is an SQL injection tool. It splits the command channel and the data exfiltration channel. The command channel can be used to create data in a database and execute other commands. It supports three data exfiltration channels: http errors, timing and DNS. For the DNS channel data is encoded in the Fully Qualified Domain Name (FQDN) used in the request (Haroon, 2007).

4.12. tcp-over-dns

tcp-overdns was released in 2008. It has a Java based server and a Java based client. It runs on Windows, Linux and Solaris. It supports LZMA compression and both TCP and UDP traffic tunneling (Analogbit, 2008).

4.13. TUNS

TUNS was developed by Lucas Nussbaum. TUNS is written in Ruby. It does not use any experimental or seldom used record types. It uses only CNAME records. It adjusts the MTU used to 140 characters to match the data in a DNS request. TUNS may be harder to detect, but it comes at a performance cost (Nussbaum, 2009).

4.14. Malware using DNS

DNS has been used as a communication method by malware. Known malware using DNS include: Feederbot (Dietrich, 2011) and Moto (Mullaney, 2011). Both of these malware examples use DNS TXT records for command and control.

5. Detection Techniques

Many of the DNS tunneling utilities do not try to be stealthy. They are relying on the fact that DNS is often not monitored. Various DNS tunnel detection techniques have been proposed. The detection techniques will be discussed as two separate categories, payload analysis and traffic analysis. For payload analysis the DNS payload for one or more requests and responses will be analyzed for tunnel indicators. For traffic analysis the traffic will be analyzed over time. The count, frequency and other request attributes will be considered.

5.1. Payload Analysis

For the payload analysis detection techniques we can also borrow from research on Domain Generation Algorithms (DGA). DGA generated domains are abnormal in a similar way to names from data encoding. The following techniques have been covered in past research.

5.1.1. Size of request and response

One technique involves analyzing the size of the request and response. In a blog post (Bianco, 2006) the author defines methods to identify suspicious DNS tunneling traffic based on the ratio of the source and destination bytes. DNS data stored in a MySQL database as part of a Snort/Squil intrusion detection system is queried for the source bytes and the destination bytes. The ratio is then compared against a limit value.

Others (Pietraszek, 2004), (Skoudis, 2012) have proposed looking at the length of the DNS query and response to detect tunneling. DNS tunneling utilities usually try to put as much data in to requests and responses possible. Thus, it is likely that tunneling requests will have long labels, up to 63 characters and long overall names up to 255 characters. Another recommendation is to look at all hostname requests longer than 52 characters (Guy, 2009).

5.1.2. Entropy of hostnames

DNS tunnels can be detected based on entropy of requested hostnames (Van Horenbeeck, 2006), (Butler, 2011). Legitimate DNS names often have dictionary words or something that looks meaningful. Encoded names have a higher entropy and a more

even use of the characters set. Although, there are exceptions to this where DNS names are used to represent some type of information. This is sometimes the case with content delivery networks. Looking for DNS names that have high entropy can be an indicator of tunneling.

5.1.3. Statistical Analysis

Looking at specific character makeup of DNS names is another method that can be used to detect tunneling. Legitimate DNS names tend to have few numbers whereas encoded names can have a lot of numbers. Looking at the percentage of numerical characters in domain names has been proposed (Bilge, 2011). Looking at the percentage of the length of the Longest Meaningful Substring (LMS) is another character makeup based method (Bilge, 2011). Looking at the number of unique characters is another possibility. A recommendation of alerting on anything with over 27 unique characters (Guy, 2009) has been provided.

Given that legitimate domain names mirror common languages to a certain extent, the use of character frequency analysis (Born, 2010b) could also be used to detect names generated from DNS tunneling.

Repeated consonants can be looked at to detect DNS tunneling, or repeated consonants and numbers (Lockington, 2012). A tunneling utility will potentially create names with consecutive consonants and numbers that you are unlikely to see in domain names that mirror common languages.

5.1.4. Uncommon Record Types

Looking for records that are not commonly used by a typical client e.g. 'TXT' records (Pietraszek, 2004) is another possible detection method.

5.1.5. Policy Violation

If a policy requires all DNS look ups to go through an internal DNS server, violations of that policy could be used as a detection method. Traffic could be monitored for DNS requests directly to the internet (Fry, 2009). Note that most DNS tunneling utilities are designed to function even when forwarding requests through an internal DNS server.

5.1.6. Specific Signatures

In some cases, researchers have provided signatures for specific DNS tunneling utilities. A signature can be used to check specific attributes in a DNS header and check for specific content in the payload. For example, a Snort signature was developed for detecting NSTX DNS tunneling (Van Horenbeeck, 2006).

```
alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"Potential NSTX DNS
Tunneling"; content:"|01 00|"; offset:2; within:4; content:"cT";
offset:12; depth:3; content:"|00 10 00 01|"; within:255; classtype:bad-
unknown; sid:1000 2;)
```

In the above example the essential parts of this rule are as follows. This signature has three content matches. The offset 2 will skip the ID part of the DNS header which represent the first two bytes. The next two bytes represent multiple header attributes. With the '01' hex, bits 0 through 6 are 0 and bit 7 is 1. This corresponds to a DNS query, not a response and the 4 opcode bits are 0 meaning it is a standard query. The 1 in bit 7 indicates that recursion is desired. The next content match begins an offset 12. This is the start of the Question section and specifically the QNAME field. Therefore, this content match is looking for "cT" as part of the first 3 bytes of the domain name. The last content match is looking for the hex values "00 10 00 01" within the first 255 bytes of the payload. This will match the two bytes of the QTYPE and QLASS parts of the Question section when the QTYPE is hexadecimal 10 (decimal 16) and the QCLASS is 1 which corresponds to IN for the Internet. IN is the only commonly used QCLASS, so it will always match the "00 01". A QTYPE of decimal 16 corresponds to TXT resource record types (Aitchison, 2011). To summarize, this signature is looking for a standard DNS query for a TXT resource record with text "cT" near the beginning of the domain name.

5.2. Traffic Analysis

Traffic Analysis involves looking at multiple requests/response pairs over time. The amount and frequency of requests can be used for an indication of tunneling. For the traffic analysis detection techniques, the following have been covered in past research.

5.2.1. Volume of DNS traffic per IP address

One basic and straight forward method is to look at the amount of DNS traffic generated by a specific client IP address (Pietraszek, 2004), (Van Horenbeeck, 2006). Because tunneled data is typically limited to 512 bytes per request, a large number of requests are required for communication. In addition, if the client is polling the server, it will continuously send requests.

5.2.2. Volume of DNS traffic per domain

Another basic method is to look at large amounts of traffic to a specific domain name (Butler, 2011). DNS tunnel utilities are all setup to tunnel the data using a specific domain name so, all tunneled traffic will be to that domain name. We should consider the possibility that DNS tunneling could be configured with multiple domain names, thus lowering the amount of traffic per domain.

5.2.3. Number of hostnames per domain

Number of hostnames for a given domain can be an indicator (Guy, 2009). DNS tunneling utilities request a unique hostname on each request. This can lead to a much larger number than a typical legitimate domain name. This method will be used as an example implementation using traffic analysis.

5.2.4. Geographic location of DNS server

Geographic considerations are another factor that could be used. As proposed, 'Large amounts of DNS traffic to parts of the world where you don't do business' (Skoudis, 2012). For enterprises that don't have a broad international footprint, this method could be useful.

5.2.5. Domain history

Domain history can also be used to raise suspicion on DNS traffic. Checking when an 'A' record or 'NS' record was added (Zrdnja, 2007). This method was used in detecting domain names involved in malicious activity. It is also relevant to detecting DNS tunneling. A domain could have been recently acquired for the purpose of DNS tunneling and its NS records could be added recently.

5.2.6. Volume of NXDomain responses

Looking for excessive NXDomain responses was proposed for detecting DGA generated names. (Antonakakis, 2012). This method could be useful for detecting Heyoka which can generate large amounts of NXDomain responses.

5.2.7. Visualization

It has been shown that Visualization can be used for detecting DNS tunnels (Guy, 2009). This method would involve interactive work by an analyst, but tunneled traffic stands out dramatically.

5.2.8. Orphan DNS requests

While most detection method look at what we can see, another approach is look for what we expect to see, but is missing. For general computing a DNS request is only made prior to another request, for example a web page request over http. With this in mind another detection method is to look for DNS requests that don't have a corresponding request by another application such as http. There will be exceptions that could easily be filtered out. Security devices may do reverse lookups on IP addresses. Anti-spam solutions use DNS queries to check if a given IP address is on a black list. An endpoint security product uses DNS queries with an encoded file hash embedded in the FQDN to check the reputation of suspicious file.

5.2.9. General covert channel detection

Some techniques for covert channel detection independent of protocol have been addressed in other research (Couture, 2010). Utilities designed for tunnel detection can look at things like request time of day or compare traffic to a statistical fingerprint.

6. Implementation of Detections

A plethora of possible detections methods have been presented. For actual implementation, the methods need to be weighed for cost and effectiveness. The cost includes hard costs such as procuring detection systems, time to develop signatures and computer resources. Given these considerations, the detections implemented in this paper are constrained to those implemented with a commercial product that will capture

and parse network traffic. While a specific commercial system was used for this paper other similar systems could be used as well. This system will be referred to generically as CAP (Capture And Parse). The system captures network traffic through the use of a TAP or span port. The captured traffic is parsed for multiple protocols including DNS. The parsed meta data can then be queried interactively with a simple rule language including limited use of Regular Expressions (regex). A reporting capability is also available which is typically used for daily reporting for traffic that matches specific rules. In addition to the built capabilities, an application program interface (API) is available for direct access to the data. This API will be leveraged for traffic analysis which can't be performed using the built in functionality.

6.1. Defense in Depth

Defense in depth is a security strategy where there are multiple layers of security. If one layer fails to detect malicious activity, another layer is in place to also detect it. By implementing rules using both payload analysis and traffic analysis some level of defense in depth is achieved.

6.2. Payload Analysis matching Fully Qualified Domain Names (FQDN)

For this paper, the DNS tunneling utilities DNScat-P and DNScat-B are chosen for analysis, implementation and testing of detection via payload analysis.

6.2.1. DNScat-P data traffic

DNScat-P uses base 64 encoding. It uses both upper case letters and lower case letter as well as the "-" and "_" in the FQDN. Looking at the source code "SixBitDNSEncoder.java", it indicates that "longer names are separated by dots (every NAMELEN characters). The class also adds encoded the frame length (the first character)" and also "NAMELEN = 30". This has been observed in DNScat-P traffic. The first label is 31 characters long and the following ones are 30 characters long. There are up to seven labels depending on data size. To match DNScat-P data traffic, the following rule will be used:

alias.host regex '^[a-zA-Z-]{31,31}[[.period.]][a-zA-Z]{30,30}[[.period.]]'

The “alias.host” keyword is the name the CAP system uses for the FQDN. The “^” character is an indicator to match at the very beginning of the FQDN. The “[a-zA-Z-]{31,31}” will match a string using any of the characters a through z or A through Z or “-“ that is 31 characters long. The “[[.period.]]” matches the “.” character. The “[a-zA-Z-]{30,30}” will match a string using any of the characters a through z or A through Z or “-“ that is 31 characters long. This signature will match an FQDN where the first label is 31 characters long and the second label is 30 characters long.

6.2.2. DNScat-P polling traffic

Many of the DNS tunneling utilities perform polling. Typically, the server can’t contact the client directly. The client therefore polls the server on a regular basis. If the server has data to send to the client, it can be sent as a response to the polling. Based on observation, the DNScat-P polling begins with a letter character, followed by a dash “-“ followed by six letter characters. The characters can be upper or lower case. To match DNScat-P polling, the following rule will be used:

alias.host regex '^a-[a-zA-Z0-9]{6,6}[[.period.]]'

The “alias.host” keyword is the name the CAP system uses for the FQDN. The “^” character is an indicator to match at the very beginning of the FQDN. The remaining part “a-[a-zA-Z]{6,6}[[.period.]” is used to match the letter “a” followed by a dash followed by six letters then a “.”.

6.2.3. DNScat-B FQDN prefix

The DNScat-B website describes the elements that make up the FQDN that it generates (Bowes, 2010). There are variations in the makeup of the FQDN depending on the mode (datagram or stream) and whether or not a session is used. The makeup for a datagram mode without a session looks like “<signature>.<flags>.<count>.<data>.<garbage>.<domain>”. For this rule, we will focus in on the signature element. This element is present in all FQDNs generated by DNScat-B. By default it is set to “dnscat” although it is easily changed using the “—signature” command line switch. To match FQDNs that begin with dnscat, the following rule will be used:

alias.host begins'dnscat'

The “alias.host” keyword is the name the CAP system uses for the FQDN. The ‘begins’ keyword means we want to look for any alias.host that begin with the following string. The ‘dnscat’ part of the rule indicates that the alias.host must begin with the string ‘dnscat’.

6.2.4. DNScat-B NetBIOS encoded traffic

The DNScat-B utility uses NetBIOS encoding by default. With this encoding type, “every character will be in the range of ‘A’ to ‘O’” (Bowes, 2010). When DNScat-B transfers data, there is by default domain labels of length 63 which can be changed using the –chunksize command line option. There is by default 3 labels used for data which can be changed by using the –sections command line option. To detect NetBIOS encoded traffic that looks for the data part of the FQDN the following rule will be used:

alias.host regex '[[.period.]][a-o]{50,63}[[.period.]][a-o]{50,63}[[.period.]]'

The “alias.host” keyword is the name the CAP system uses for the FQDN. The “[[.period.]]” matches the “.” character. The “[a-o]{50,63}” look for a string using any of the characters a through o that is 50 to 63 characters long.

6.2.5. DNScat-B Hex encoded traffic

The DNScat-B utility can use hexadecimal encoding by using the –encoding command line option. With this encoding method, each byte is represented by two hexadecimal characters. The results is that the characters in the FQDN will be limited to 0 through 9 and a through f.

To detect Hexadecimal encoded traffic that looks for the data part of the FQDN he following rule will be used:

alias.host regex '[[.period.]][0-9a-f]{50,63}[[.period.]][0-9a-f]{50,63}[[.period.]]'

The “alias.host” keyword is the name the CAP system uses for the FQDN. The “[[.period.]]” matches the “.” character. The “[0-9a-f]{50,63}” look for a string using any of the characters 0 through 9, a through f that is 50 to 63 characters long.

6.2.6. Generic DNS tunnel detection

DNS tunneling utilities typically try to transfer as much data as possible in a request. This typically means long FQDNs approaching the limit of 255 as well as long label names. Therefore we can make a generic version of the data rules used for DNScat-B and DNScat-P. For this rule we will allow any characters and look for three labels instead of two.

alias.host regex
'[[.period.]][^.]{25,63}[[.period.]][^.]{25,63}[[.period.]][^.]{25,63}[[.period.]]'

The first matched character must be a "." which can occur anywhere within the FQDN. The "[^.]{25,63}" matches a string that is 25 to 63 characters long made up of characters that are not ".". The remaining part of the rule repeats the previous parts. The end result is to match three consecutive labels that are between 25 and 63 characters.

6.3. Traffic Analysis

Some of the specific payload analysis rules could be bypassed by a knowledgeable attacker. They could shorten the labels used and reduce the number of labels in an FQDN. However, they will always need to create a large number of unique FQDNs which are normally from a specific root domain. The root domain is the two right most labels namely the second level domain and the top level domain. For the FQDN "www.local.example.com" the root domain is "example.com". Multiple root domains could be used for tunneling but, this would still likely result in a large number of FQDNs per root domain. Therefore detection on the number of FQDNs per root domain would be a very valuable rule. The CAP system in use does not provide a method to report on number of FQDNs per root domain. It does however have an Application Program Interface or API which can be used to access the stored data and create the desired rule.

6.3.1. API overview

The CAP system uses a Representational state transfer or (REST) API via http/https. Through the API stored data can be requested and received in text, html, xml

or json format. A prototype was developed using Python and the nwmodule library from nwmaltego (Bressler, 2012).

6.3.2. Top Number of Fully Qualified Domain Names per root domain

The prototype traffic analysis rule developed with the API is a multi-step process. These steps are all included in a Python script which collects and analyzes DNS data. First, DNS data for a specific time window is accessed via the API and saved as an xml file. Second, the xml file is cleaned up to prepare for import. Third, the xml file is imported into mysql. Fourth, the data are analyzed using Python and mysql to find the top counts of unique FQDNs per root domain. These steps have additional details of interest.

Step 1: The key parts of this step are the query to select data and the output type. This line (note: line is wrapped) shows the data selection:

```
qstring = "select time, ip.src, ip.dst, alias.host where
service='53' && monitor='external' && time='" + texttimestart + "'-'" +
texttimeend+"'  && tld != 'arpa'"
```

Time, source IP address, destination IP address and FQDN are selected for DNS traffic over a given time range. Only external DNS queries are considered to reduce the amount of processing. Also, domains ending in ".arpa" are excluded to remove reverse lookups from consideration.

The next two lines show the query and output type.

```
ctype="text/xml"

nwquery = nwmodule.nwQuery(0, 0, qstring, ctype, 1000000)
```

The output type is xml. The next step is to clean up the xml

Step 2: The xml output is not ready for import in to xml. One of the fields uses the name 'group' which is a reserved word in mysql. Using 'group' for a column header is problematic so it will be changed to 'session'. The other problem is that the last data is not presented with an attribute name. The cleanup is best presented with an example. For the example only one line is shown. A session will typically have additional lines for source IP, destination IP and time.

Line before cleanup:

```
<field count="0" flags="0" format="65" group="18144159523"
id1="658135654510" id2="658135654510" size="18"
type="alias.host">www.example.com</field>
```

Line after cleanup:

```
<field count="0" flags="0" format="65" session="18144159523"
id1="658135654510" id2="658135654510" size="18" type="alias.host"
data="www.example.com"> </field>
```

Step 3: Once the file is cleaned up, import in to mysql is relatively straight forward. The following line is used:

```
LOAD XML INFILE 'CAPexport.xml' INTO TABLE dnsx rows identified
by '<field>';
```

After the import is complete the table is available in mysql. Note that a given session has multiple rows in the table and that the data column has different types of values in it which can be time, source IP address, destination IP address or FQDN. There are often multiple FQDN lines with the FQDN repeated. The data will be reorganized when they are analyzed.

Step 4: The data are analyzed using the Python script to eventually report on root domains with the highest number of unique FQDNs. This step has two sub steps. First, there is a loop performed on distinct session ids. For each unique session, a row is created in a new table which has columns for each data type: time, source IP, destination IP and FQDN. Additionally, the root domain is extracted and added in its own column. Next, there is a loop on the new table on distinct root domains. For each one a count of unique FQDNs is reported. This is the desired end result. By looking at the FQDN count, DNS tunnels can be identified by having a very high number of unique FQDNs for a given domain. This is best illustrated by looking at some example test data.

6.3.3. Example Test Data

For the test data, traffic was monitored for one day during normal business hours. Over 380,000 external DNS queries occurred during the business day. The environment has over 1,000 users. A DNS Tunnel was created using DNScat-B at one point during

the day and used to transfer a 122K text file (the text of RFC 1035). The file transferred in about 2 minutes and the tunnel remained up for a few minutes after that. The 'Unique FQDN count per root domain' detection technique described in 5.3.2 was run for sample windows of ten minutes each.

Sampling windows longer than 10 minutes can cause the processing time to be longer the sampling window. A shorter window is also more sensitive to short duration DNS tunneling because non tunnel traffic will have accumulated fewer unique FQDNs. Even with larger sampling windows, the DNS tunneling activity can be easily identified if white listing is included for known sites that have more than 100 unique FQDNs.

Figure 1 shows a timeline of the DNS traffic from the tunneling activity. The vertical axis is number of sessions.

DNS Tunnel traffic timeline

Figure 1.

The straight line at the end of the graph occurs because the file completed transfer and the tunnel went in to polling mode where a constant amount of requests are sent to the DNS tunnel server.

In the tables below, the normal root domain names were replaced with single letter labels. The root domain used for tunneling was replaced with the label 'TUN'. Figure 2 shows the ten minute window just before any tunneling traffic occurs. The vertical axis is number of unique FQDNs. Domains that have a high number of unique FQDNs fall in to a few different categories. Domains used for advertising, content delivery and cloud services often use a large number of unique FQDNs. Domains providing a wide variety of services can have a large number of FQDNs. One interesting example is label 'A'. This domain is a security service that uses encoded file hashes as part of an FQDN to check a file's reputation. This domain has a large number of unique FQDNs. One other example is a second level domain that is used to provide domains for other organizations, for example 'co.uk'. This type could be accounted for by consider

the third level domain as part of the root domain. The domains in the figures below all fall in one of the above categories or the actual tunnel traffic.

Time window of 10 minutes before tunnel traffic

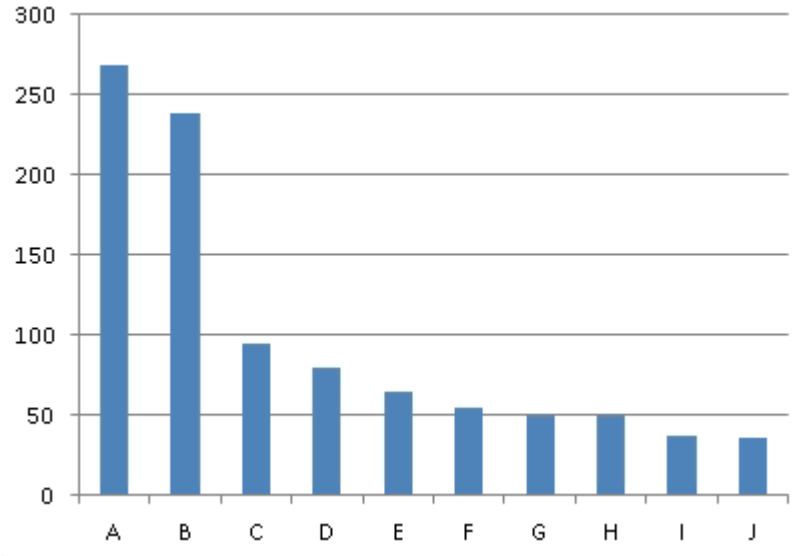

Figure 2.

Typically as shown in Figure 2, all domains have fewer than 300 unique FQDNs for a ten minute sample window. A threshold of 300 could be used to alert on potential DNS tunneling domains.

Time window of 10 minutes during tunnel traffic

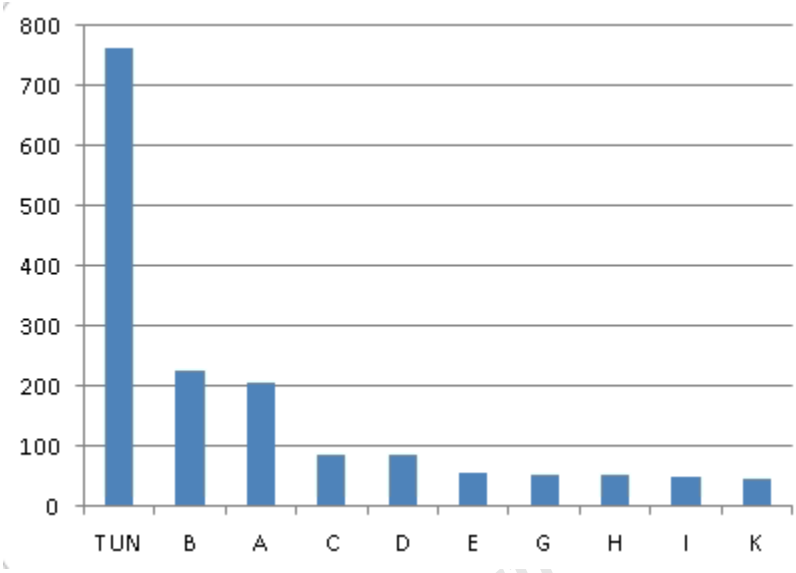

Figure 3.

Figure 3 is the ten minute window in which DNS tunneling was used to exfiltrate a 122K file. As you can see the column with label 'TUN' is significantly more than all the others. The domain labeled with 'TUN' is the one used for the DNS tunneling. This shows that tunneling can be detected by using a count of unique FQDNs per root domain.

Time window of 10 minutes during tunnel polling traffic

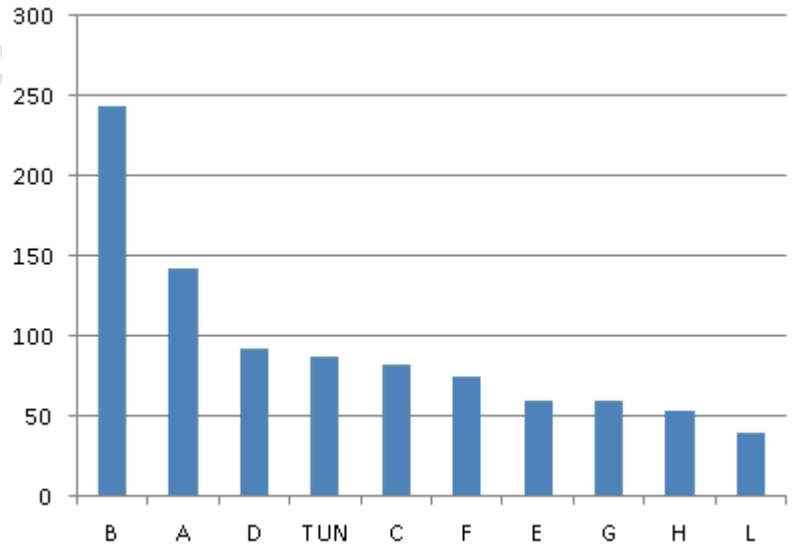

Figure 4.

Figure 4 shows the ten minute window after the file transfer. There is still DNS traffic to domain 'TUN'. This is from the polling activity of DNScat-B. In this time window, the level of activity of the tunnel is much lower as no data is being transferred. Even with this lower level of activity, the 'TUN' root domain has one of the top four numbers of unique FQDNs.

Time window of 1 hour including DNS Tunneling

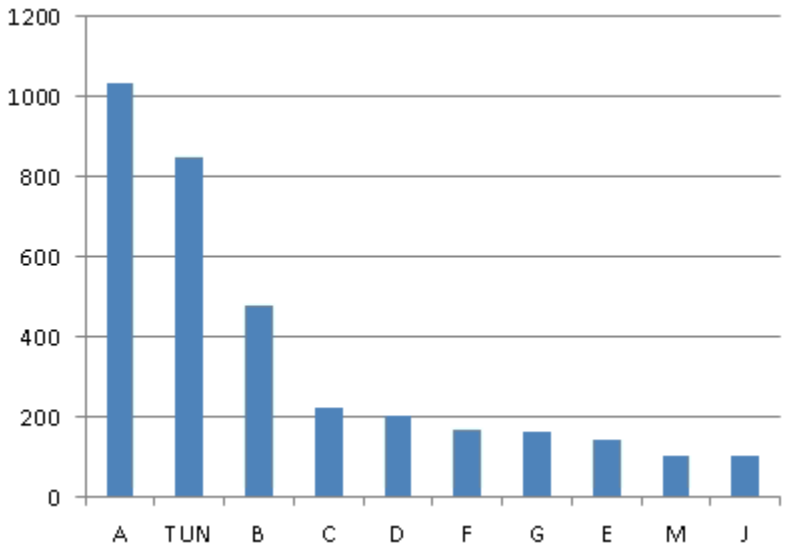

Figure 5.

Figure 5 shows the number of unique FQDNs for a 1 hour sampling window. The tunneling traffic 'TUN' is only eclipsed by label 'A' which is the file reputation security service.

Time window of 9 hours including DNS Tunneling

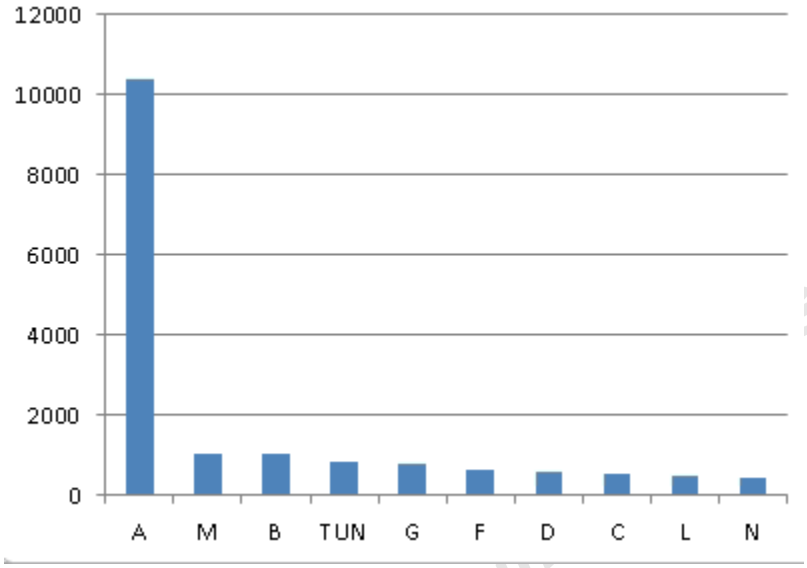

Figure 6.

Figure 6 shows the number of unique FQDNs for a complete work day (8am-5pm local time). The 'A' label is significantly higher than the others. This is expected since for this security service each request for a unique file will have a unique FQDN. The domain used for the tunneling is the fourth highest even though it was from a few minutes of tunneling compared to 9 hours of activity for other domains. Again, as mentioned earlier, white listing could be used to remove the top sites from normal traffic making even a few minutes of DNS tunneling stand out even when looking at a full day of data.

7. Conclusion

A large number of DNS Tunneling utilities exist with a wide range of capabilities. They provide a covert channel for malicious activities which represent a significant threat to organizations. These threats can be mitigated using payload analysis and traffic analysis detection techniques.

There are over a dozen different DNS tunneling utilities covered in this paper. They range from netcat like capability over DNS to full IP tunneling over DNS. While some utilities have been available for several years, others have been recently developed

with improved capability. For example, Heyoka uses source spoofing to lower the visibility of the compromised endpoint. Additionally, DNS tunneling capability is conveniently available as part of penetration testing tools Metasploit and squeeza.

DNS tunneling represents a significant threat to organizations. The two main threats of DNS tunneling are command and control of compromised endpoints and data exfiltration. DNS tunneling can be used for command and control as has seen with Feederbot and Moto malware. Command and control can also include full remote access of a compromised endpoint. This can be accomplished using netcat like capability with some DNS tunneling utilities or via any remote access application over a full IP tunnel. The other threat is data exfiltration. DNS tunneling provides a covert channel for data exfiltration. Although inefficient for data transfer DNS tunneling can be used to easily exfiltrate high value data such as password hashes or sensitive documents. If left unmonitored, DNS tunneling can be used to exfiltrated large amounts of data given enough time.

The threat of DNS tunneling can be mitigated by implemented payload analysis and traffic analysis techniques. Payload analysis can be used to detect DNS tunneling using signatures based on attributes of individual DNS payloads such as the FQDN contents. Payload analysis is most effective for detecting known DNS tunneling utilities. Traffic analysis is the other detection technique. Traffic analysis can be used to detect DNS tunneling based on characteristics of overall traffic. Using traffic analysis, a universal DNS tunneling detector can be implemented. This is achieved by monitoring the count of unique FQDNs for a give root domain. This technique is independent of DNS resource record type, encoding, DNS label length and FQDN length. This technique was successfully demonstrated in this paper; however, there is room for improvement. The 'count of unique FQDNS' method could be improved by optimizing the components and algorithms to reduce the resource requirements. It could also be improved by adding features to account for white listing of known root domains with a high count of unique FQDNs. The practical detection methods provided in this paper can be used to successfully detect DNS tunneling. With these detection methods in place organizations can reduce the risk associated with DNS tunneling.

8. References

Aitchison, R. (2011). *Pro ns and bind 10*. (p. 493). New York, NY, USA: Apress.

Analogbit. (2008, July 27). tcp-over-dns. Retrieved from http://analogbit.com/software/tcp-over-dns

Andersson, B. (2010). *iodine by kryo*. Retrieved from http://code.kryo.se/iodine/

Antonakakis, M. (2012). *Dgas and cyber-criminals: A case study*. Retrieved from https://www.damballa.com/downloads/r_pubs/RN_DGAs-and-Cyber-Criminals-A-Case-Study.pdf

Bianco, D. (2006, May 3). A traffic-analysis approach to detecting dns tunnels. Retrieved from http://blog.vorant.com/2006/05/traffic-analysis-approach-to-detecting.html

Bienaime, P. (2011). *dnscapy, dns tunneling with scapy*. Retrieved from http://code.google.com/p/dnscapy/

Bilge, L. (2011). *Exposure: Finding malicious domains using passive dns analysis*. Retrieved from http://www.syssec-project.eu/media/page-media/3/bilge-ndss11.pdf

Born, K. (2010a). *Psudp: A passive approach to network-wide covert communication*. Retrieved from http://www.kentonborn.com/sites/default/files/psudp_born_slides_bh_2010.pdf

Born, K. (2010b). *Dns tunnel detection using character frequency analysis*. Retrieved from http://www.kentonborn.com/sites/default/files/dns_cfa.pdf

Bowes, R. (2010, February 2). *Dnscat*. Retrieved from http://www.skullsecurity.org/wiki/index.php/Dnscat

Bressler, D. (2012, December). *nwmaltego*. Retrieved from https://github.com/bostonlink/nwmaltego

Butler, P. (2011). *Quantitatively analyzing stealthy communication channels*. Informally published manuscript, Computer Science, Virginia Tech, Blacksburg, VA, .

Couture, E. (2010, August 19). *Covert channels*. Retrieved from http://www.sans.org/reading_room/whitepapers/detection/covert-channels_33413

Dembour, O. (2008, November 3). Dns2tcp. Retrieved from http://www.hsc.fr/ressources/outils/dns2tcp/index.html.en

Dietrich, C. (2011, September 2). *Feederbot - a bot using dns as carrier for its c&c*. Retrieved from http://blog.cj2s.de/archives/28-Feederbot-a-bot-using-DNS-as-carrier-for-its-CC.html

Fry, C. (2009). *Security monitoring, proven methods for incident detection on enterprise networks*. (1st ed., p. 28). Sebastopol, CA, USA: O'Reilly Media.

Guy, J. (2009, February 13). *dns part ii: visualization*. Retrieved from http://armatum.com/blog/2009/dns-part-ii/

Haroon, M. (2007, August). *Squeeza: Sql injection without the pain of syringes*. Retrieved from http://www.sensepost.com/cms/resources/labs/tools/pentest/squeeza/READMEv0.21.txt

Kozierok, C. (2005). *The TCP/IP Guide*. (p. 849). San Francisco, CA, USA: no starch press.

Lockington, S. (2012, January 16). *Detecting the bad from the good*. Retrieved from http://scottfromsecurity.com/blog/2012/01/16/detecting-the-bad-from-the-good/

mdornseif. (2002, August 03). Denise. Retrieved from https://github.com/mdornseif/DeNiSe

Mullaney, C. (2011, August 31). *Morto worm sets a (dns) record*. Retrieved from http://www.symantec.com/connect/blogs/morto-worm-sets-dns-record

Nussbaum, L. (2009). *On robust covert channels inside dns*. Retrieved from http://hal.inria.fr/docs/00/42/56/16/PDF/tuns-sec09-article.pdf

NSTX. (2002, September 22). NSTX -- tunneling network-packets over dns - summary. Retrieved from http://savannah.nongnu.org/projects/nstx/

Pearson, O. (1998, April 13). *Dns tunnel - through bastion hosts*. Retrieved from http://archives.neohapsis.com/archives/bugtraq/1998_2/0079.html

Pietraszek, T. (2004, October 31). Dnscat. Retrieved from http://tadek.pietraszek.org/projects/DNScat/

Rasmussen, R. (2012, April 03). *Do you know what your dns resolver is doing right now?*. Retrieved from http://www.securityweek.com/do-you-know-what-your-dns-resolver-doing-right-now

Revelli, A. (2009). *Playing with heyoka*. Retrieved from http://heyoka.sourceforge.net/heyoka-shakacon2009.pdf

Skoudis, E. (2012, February 29). The six most dangerous new attack techniques and what's coming next?. Retrieved from https://blogs.sans.org/pen-testing/files/2012/03/RSA-2012-EXP-108-Skoudis-Ullrich.pdf

Kliarsky, A. (2010, August 19). *Covert channels*. Retrieved from http://www.sans.org/reading_room/whitepapers/detection/covert-channels_33413

Van Leijenhorst, T. (2008). *On the viability and performance of dns tunneling*. Retrieved from http://www.uow.edu.au/~kwanwu/DNSTunnel.pdf

Van Horenbeeck, M. (2006). Dns tunneling. Retrieved from http://web.archive.org/web/20060613210141/http://www.daemon.be/maarten/dnstunnel.html

Vixie, P. (1999, August). *Extension mechanisms for dns (edns0)*. Retrieved from http://www.ietf.org/rfc/rfc2671.txt

Wong, M. (2006, April). *Sender policy framework (spf) for authorizing use of domains in e-mail, version 1*. Retrieved from http://tools.ietf.org/html/rfc4408

Zdrnja, B. (2007). *Passive monitoring of dns anomolies*. Retrieved from http://www.caida.org/publications/papers/2007/dns_anomalies/dns_anomalies.pdf