A Novel Approach to Detecting Covert DNS Tunnels Using Throughput Estimation

Michael Himbeault

May 27, 2012

Abstract

In a world that runs on data, protection of that data and protection of the *motion* of that data is of the utmost importance. Covert communication channels attempt to circumvent established methods of control, such as firewalls and proxies, by utilizing non-standard means of getting messages between two endpoints. DNS (Domain Name System), the system that translates text-based resource names into resource records, is a very common and effective platform upon which covert channels are often built. This work proposes, and demonstrates the effectiveness of, a novel new technique that estimates data transmission throughput over DNS in order to identify the existence of a DNS tunnel. The proposed technique is robust in the face of the obfuscation techniques that are able to hide tunnels from existing detection methods.

Contents

1 Introduction

Because it is important to control the data that is sent between hosts, systems such as firewalls, proxies, and content filters are put into place in order to monitor and control the network traffic. Covert channels, such as DNS tunnels, are utilized to circumvent these control mechanisms for purposes that range from benign to malicious. This work focuses specifically on DNS tunnels, however other types of covert channels do exist.

Detecting a DNS tunnel effectively on a busy network link becomes an exercise in discrimination. Because there is such a wide variety of network traffic that is generated on a busy link, there is generally no simple definition of *normal* for a particular class of traffic, including DNS. This tends to either rule out, or decrease the viability of, algorithms that depend on finding a definition of normal and alerting based on deviations from that norm.

A highly sensitive and specific detection of DNS tunnels on a busy network link is a very important problem in the arena of network security as it enables administrators to block or otherwise control these potential sources of compromise. A non-exhaustive, but informative, list of things that DNS tunnels can be used for is:

- Data exfiltration
- Two-way communication
- Communication through restrictive firewalls
- Transport layer for complete VPN solutions
- Arbitrary data transmission

In enterprises that deal with sensitive information such as financial, health, personal or intellectual property information it is of the utmost importance to control the access to this information. Even if an enterprise does not have information that needs protecting, DNS tunnels should still be blocked in order to prevent malware from communicating. Because DNS tunnels can be used for arbitrary communication, they can be used as command-and-control channels for botnets or any other malicious system that relies on data transmission.

Preventing botnets from operating is of the best interest for the Internet as a whole and should be a concern of every user of the Internet.

Existing methods of detection rely on one of two methods, character frequency and signatures, to detect the presence of a DNS tunnel. The signature-based solutions attempt to identify portions of the tunneling application data (as opposed to the user data that is sent using the tunnel application) for which a signature can be built. These signature based solutions are subject to many of the same problems that plague signature based anti-virus solutions. For example, if the application changes its communication schemes, the signatures may no longer be valid and may no longer trigger when they are intended to. Additionally, signatures cannot be built for applications that are not available for study, or that aren't known of. For applications that use a custom communication scheme that has not had a signature built specifically for it, it is very likely that an existing signature will not be relevant and will not detect it.

Because signature based schemes are not effective in detecting DNS tunnels in a zero-day[1] situation, another method is required. Analysis based on character frequency is proposed in[**?**] by Kenton Born which uses character frequency analysis under the assumption that normal DNS traffic has a unique character frequency distribution that can be used as a discriminating fingerprint. Based on analysis of common domain names, a distribution is obtained that accurately describes normal DNS traffic while effectively detecting DNS tunnel traffic. Results obtained by Born indicate that DNS tunnels have an approximately uniform character frequency distribution whereas normal DNS queries do not.

A weakness of the approach proposed by Born is that it relies on the assumption that DNS tunnel traffic *necessarily* has a character frequency distribution that is different from that of normal DNS queries. This assumption is not necessarily true, and a proof-of-concept was developed that demonstrates this fact. The tool performs a *probabilistic encoding* that takes an arbitrary data source and encodes it into a stream of characters that conforms to a

[1] *Zero-day* refers to a situation where nothing is known about the attack as it has never been seen or analyzed before.

given distribution. This tool can be used to modify the output of any DNS tunnel application so that their output conforms to the distribution that Born found for normal DNS traffic. By utilizing this transformation, Born's approach fails to detect the DNS traffic as it becomes, from the viewpoint of his algorithm, indistinguishable from normal DNS traffic. Details of this are given in **??**.

Due to the weaknesses listed above, it is clear that a new approach is necessary that detects DNS tunnels in a zero-day situation and without the known weakness involving character frequency and probabilistic encoding. This work proposes, and demonstrates the effectiveness of, a method of detecting DNS tunnels that meets these requirements.

The proposed method operates under the assumption that DNS tunnels move more data than a normal domain. By leveraging this fact, it is possible to detect any use of DNS to transmit arbitrary data by measuring the amount of data that is transmitted using a particular DNS domain or subdomain. The details of this measurement methodology is contained in **??** and requires estimating the amount of unique data that is transmitted by analyzing the queries themselves and not simply counting characters in the query string.

This proposed detection methodology is shown to detect DNS tunnels in as few as ten packets and continues to be robust in highly hostile detection environments such as those that contain a great deal of non-tunnel traffic as well as benign uses of DNS for transmitting arbitrary data[2]. Detector performance on commodity hardware is shown to scale to greater than two gigabits of UDP port 53 throughput per second, indicating that this methodology does not sacrifice performance for detection accuracy and remains practical for monitoring very large networks.

[2]Many security vendors utilize the fact that DNS and UDP port 53 are so loosely controlled in order to ensure that their deployed devices can communicate with the vendor intelligence unimpeded.

2 Background

2.1 Domain Name System (DNS)

DNS (Domain Name System) is the service through which names are mapped to resources. Typically, this maps a name (such as *www.google.ca*) to an IP address. The value of this service is that names are considerably more flexible, and considerably easier to remember, than the resource or record that they point to. For example, *google.com* is considerably easier to remember than one of the IP addresses that it points to, such as 74.125.226.34. *google.com* is also considerably more flexible, since it points to not just one but several addresses, and successive responses will receive different records in a round-robin, or random, fashion. This rotation of responses allows for a crude form of load balancing and automatic fail over, while retaining the ease of use.

The DNS protocol is assigned both UDP and TCP port 53 for communication with most communication operating over UDP as opposed to TCP. The use of TCP depends on the implementation of the resolver, however the specification indicates the TCP should be used if the response data exceeds 512 bytes or during a zone transfer[**?**]. DNSSEC (Domain Name System Security Extensions), due to the fact that it requires a signature of authenticity for all responses, will often cause the response to require TCP[**?**]. Because there is no *requirement* for when TCP be used, some resolvers may be implemented to use TCP for all responses as this does not violate the specifications for DNS.

The experiments related to this work do not consider the situation of DNS over TCP since the analysis techniques are identical since the formats of the UDP and TCP responses is identical once the TCP stream is reassembled. Modification of the tools developed for this work would require the ability to perform TCP stream reassembly in order to extract the responses from the TCP response.

Because DNS is such an integral component of Internet communication it is not reasonable to simply block it while still expecting a functional Internet browsing experience. A common approach is called DNS *proxying* which forces all DNS queries to be made to a DNS proxy

server that is controlled by the interested entity (ISP, company, etc...). This DNS proxy server is responsible for handling all DNS queries for the internal network, and any DNS queries that are destined for the Internet are typically dropped by the firewall in this type of configuration. The DNS proxy server operates in *recursive mode*, which means that if a question is asked of it to which it does not know the answer, the proxy server will then query for the answer (by issuing its own query to the global DNS system) and then respond to the initial request using this response.

DNS is a heavily cached protocol due to how often data can be reused between queries. Consider how often a desktop Internet user causes a request for *google.com*. If a request had to traverse the entire DNS system every time, this would represent a very considerable amount of traffic being generated. To avoid this, every DNS record has extra information about it that includes, among other things, how long that it can be cached for. Standard caching lengths put it around one hour which means that a DNS server will only recursively pass on a query for a record once an hour. This caching period is not constant and can be set differently, depending on the information the record contains. Some records require a considerably lower TTL (Time To Live), as low as one minute, while for others a considerably longer (months) may be appropriate.

This proxy architecture removes some naive operation modes for DNS tunnels that will be discussed in in more detail in 2.3, but does not offer any protection against the more common forms of DNS tunnels.

2.2 Covert Channels

Covert channels are methods of communication that use non-standard means of communication for the purpose of evading detection and/or blocking by the existing security infrastructure. Covert channels may utilize portions of an existing protocol or communication channel, or they may find ways of transporting information utilizing a completely new medium. An example of the latter is called a *timing channel*, which can utilize the timing between packets to convey information. A timing channel carefully controls the timing between packets sent

to a remote server to encode information, thereby utilizing a method of communication that is not utilized by any *legitimate* protocol or communication method.

Covert channels come in may forms and not all types support the properties that are normally associated with a communication channel. Because they are built on unorthodox, or unreliable, transmission media and are subject to the effects of intermediate routing and networking devices they cannot always offer all of the same functionality as a legitimate channel.

Covert channels need not support bidirectionality due to either the constraints of the underlying medium, or the effect of intermediate devices. A covert channel that is only useful for reception is one that utilizes a third-party image hosting service. It is possible to embed arbitrary information into the header portion of an otherwise completely benign JPEG image file which could then be posted to Facebook, Flickr, or any other publicly accessible image hosting service. This image file can then be checked by the remote hosts to pull the information however, due to the nature of the image services, the remote hosts may have no way of posting information back to the other endpoint, making the communication channel unidirectional.

Real time data transfer refers to the ability for a communication channel to send data immediately. UDP, by its very nature, supports this and TCP supports this via the PUSH flag which indicates that data is being sent before a full window has been accumulated. The TCP PUSH flag is used, for example, during an SSH connection in order to provide interactivity. Timing channels, or any channel that relies on modifying normal system traffic instead of generating their own traffic, by their nature, are unable to support real time data transfer. This is because they need to wait for a system packet in order to send their data, and if the system goes for a period of time without sending data then the covert channel must wait as well.

2.3 DNS Tunnels

DNS tunneling is the method by which arbitrary data is made to be transferred over the same channels as DNS. DNS tunnels come in one of two types: raw, or conforming.

Raw DNS tunnels do not attempt to mimic or conform to the DNS specifications, and simply attempt to utilize the fact that UDP port 53 is often left open in firewalls. Raw tunnels attempt to exploit this by transmitting arbitrary traffic using UDP port 53 packets with arbitrary payload. This is the most efficient exploitation of the ubiquity of DNS as it incurs the lowest amount of overhead, both computationally and in terms of network throughput. The trade off for this efficiency is that it is the least conforming and the most likely to get stopped by either a firewall or a proxy. In the situation where all DNS queries are forced to be proxied through a dedicated DNS server, raw DNS tunnels will fail to operate as expected. This is because when the UDP port 53 traffic is redirected to the proxy, the DNS server will attempt to interpret the arbitrary payload as a DNS packet and will likely fail. When it fails, it will drop the packet thereby preventing all raw UDP port 53 communication. Because these types of tunnels are effectively blocked by standard firewall and proxy practices, detection of these tunnels is not considered in this work.

Conforming DNS tunnels produce DNS packets that conform to all appropriate specification and RFC documents and, as far as any DNS server, is concerned the traffic generated is valid DNS traffic. These tunnels incur the highest computational and throughput overhead, but have the advantage that detecting and blocking them is a very difficult process. The detection of this type of DNS tunnels is the topic of this work. This type of tunnel is capable of operating, even in an environment with very strict firewall and proxy policies. Because this type of tunnel operates in very hostile (to the operation of the tunnel) environment, detection of this type of DNS tunnel is of interest to all levels of government and industry.

Conforming DNS tunnels operate by embedding the data for transmission into the query string and the response and require a modified, non-conforming, DNS server on one end of the connection and a piece of software on the client end. Typically these types of DNS tunnels have one endpoint that is controlled by the tunnel user, with that controlled endpoint running

dedicated server software that. The client and server software is responsible for transforming arbitrary information into DNS queries, and for translating DNS responses back into the arbitrary data that it represents. The precise details of how the translation is done between DNS and the raw data depends on the implementation.

2.3.1 DNS Tunnel Software

Some DNS tunneling software is OzymanDNS[3], Iodine[4], Dns2tcp[5], DNScat[6] and DeNiSe[7], and PSUDP[8]. Each of these have slightly different operational characteristics, but they all aim to do the same thing, which is transmission of arbitrary data over DNS.

Iodine supports raw tunnels, as well as moving information back from the server in A (IPv4 addresses), MX (mail server), TXT (arbitrary text) and may other supported DNS record types.TXT records are rarely used by consumers or end-user applications, and so a blanket block policy of TXT records for end-user devices would have very little impact on end-user applications. TXT records are as close to a raw tunnel that a conforming tunnel can get to in terms of throughput.

DNScat utilizes CNAME (alias to another record) records and a supplementary A record, when appropriate, however the A record is not actually used for throughput. OzymanDNS and DeNiSe utilize solely the TXT record, which is as close to the raw tunnel as possible, however can be easily neutered by simply blocking TXT records. Because these tools only uses TXT records it is possible that it is the least flexible and deployable out of those listed above given a hostile environment. Dns2tcp utilizes with TXT or KEY records, which makes it as flexible as OzymanDNS and DeNiSe. The KEY DNS record was designated for specific uses[**?**], but has been deprecated now[**?**] in favour of the DNSKEY record for use with DNSSEC[**?**] and the IPSECKEY for use with IPSEC[**?**]. Because of this deprecation, use of the KEY record

[3] http://dnstunnel.de/
[4] http://code.kryo.se/iodine/
[5] http://hsc.fr/ressources/outils/dns2tcp/index.html.en
[6] http://tadek.pietraszek.org/projects/DNScat/
[7] http://c0re.23.nu/c0de/snap/DeNiSe-current.tar.bz2
[8] http://www.kentonborn.com/psudp

is subject to strict filtering which greatly reduces the effectiveness of this solution.

All of the above tools utilize encoding and decoding mechanisms that would successfully propagate through a proxy server, at the cost of the fact that the tunnel applications have to generate their own traffic. PSUDP, proposed by Kenton Born, aims to remove the latter requirement by creating slack space within an existing DNS packets at the UDP transport layer. He proposes two ways of creating this slack space: naively placing it at the end of the packet, and rearranging the DNS query string to utilize pointers to create this slack space in the middle of the packet. Pointers allow for the re-use of DNS query strings within a packet to save on space. They are an optional component of the specification, but allow for considerable space savings. A pointer in a DNS packet is a special sequence of bytes that indicates where in the packet the processing of the query string should jump to. When processing a query string, only a single pointer can be followed, according to spec, which prevents multiple redirection and infinite loops (where a pointer points to itself). By having a pointer point forward in the packet, it is possible to cause the parsing of a query string to skip a number of bytes, creating slack space.

This method relies on this slack space, which is not parsed by normal servers or clients, but can be contain arbitrary data that is extracted by special clients. However, because this slack space is not processed by DNS servers, in an environment where all DNS queries must go through a proxy, this method is incapable of producing a covert channel that is able to penetrate a strictly proxied DNS environment.

3 Problem Statement

DNS tunnel detection is a complicated task made worse by the fact that DNS tunnel traffic can be completely legitimate, conforming network traffic. It need not violate any established standards or conventions, which makes it difficult to detect against the background of normal DNS traffic.

This property of DNS tunnels makes them a commonly used transport mechanism when

data exfiltration or network control circumvention is the end goal. For this reason an efficient method of detecting DNS tunnels is required that can effectively detect a DNS tunnel against normal DNS traffic with a low false-positive rate and that must not be susceptible to existing methods of circumvention.

4 Review of the State of the Art