DNS Tunneling for Network Penetration

Daan Raman[1], Bjorn De Sutter[1], Bart Coppens[1], Stijn Volckaert[1], Koen De Bosschere[1], Pieter Danhieux[2], and Erik Van Buggenhout[2]

[1] Computer Systems Lab, Ghent University, Belgium
bjorn.desutter@elis.ugent.be
[2] Ernst & Young, ITRA FSO, Belgium

Abstract. Most networks are connected to the Internet through firewalls to block attacks from the outside and to limit communication initiated from the inside. Because of the limited, supposedly safe functionality of the Domain Name System protocol, its traffic is by and large neglected by firewalls. The resulting possibility for setting up information channels through DNS tunnels is already known, but all existing implementations require help from insiders to set up the tunnels. This paper presents a new Metasploit module for integrated penetration testing of DNS tunnels and uses that module to evaluate the potential of DNS tunnels as communication channels set up through standard, existing exploits and supporting many different command-and-control malware modules.

Keywords: domain name system, tunneling, Metasploit, network penetration.

1 Introduction

Private computer networks are under constant attack. Sometimes attackers try to setup so-called bind connections externally, with which they try to connect to listening sockets in the local network [20]. In other attacks infected local computers or malicious insiders try to set up reverse connections to an external computer [20]. The StuxNet command-and-control (C&C) malware was injected into a local network through a USB key, after which it set up connections to the home network to receive further commands and to leak private data [21]. Botnets are another well known type of C&C malware.

Two common methods to protect against such attacks are firewalls [1] and Intrusion Detection and Prevention Systems (IDPS) [8]. Firewalls permit or block network traffic based on protocols, blacklists and whitelists. IDPS are based on rule sets that describe suspicious network behavior. All network traffic is scanned by an IDPS, and as soon as a sensor picks up suspicious actions, the actions are interrupted and the network administrators are informed.

One commonly used network protocol is DNS, or Domain Name System [15,16]. The main functionality provided by DNS servers is to translate computer names such as www.icisc.org to the IPv4 address 164.125.70.63. This enables people and configuration files to rely on names instead of hard to remember addresses. Most

T. Kwon, M.-K. Lee, and D. Kwon (Eds.): ICISC 2012, LNCS 7839, pp. 65–77, 2013.

often, a local network has one local DNS server. When some client in the local network needs an address translation, he queries the local DNS server with a so-called lookup request. The server knows which external DNS servers to access for information on external addresses in different domains, and forwards the queries. When the contacted external DNS server is authorized for the requested (domain) name, it returns the response to the query. If that server is not authorized, it returns a link to another DNS server, that is in turn queried by the local server. This iterative process continues until an authoritative server returns the final answer. An important property is that DNS servers can only respond to queries with appropriate responses. They can, in other words, not initiate any communication themselves, and the type of any response they provide must correspond to the type of the lookup.

On the one hand, the DNS protocol thus provides very limited services. For this reason, firewalls or IDPS in practice rarely check or filter DNS traffic originating from the local DNS server. On the other hand, the iterative nature of DNS can be used to hide connections with malicious DNS servers behind connections with the local server and with external, legitimate servers.

This potential for abuse has been exploited by so-called DNS tunnels [2,6,11,12], in which local computers exchange information with malicious DNS servers through a form of steganography. The local computer transmits information by embedding it in the domain names for which it queries the malicious DNS server, and the server transmits information by encoding it in the responses it returns, such as domain name aliases or textual descriptions of properties. To the best of our knowledge, all existing implementations require explicit insider cooperation to set up a DNS tunnel, and support only limited forms of communication.

In this paper, we present a staged attack we developed in the popular Metasploit Framework (MSF) for penetration testing [19]. This attack can leverage almost all known software vulnerabilities for which exploits are present in MSF. Furthermore, it supports the installation of C&C modules in MSF through a DNS tunnel, as well as tunneling those modules' traffic through the tunnel. With this proof-of-concept attack, we demonstrate for the first time that DNS tunnels are a threat even when all legitimate users of a network are benign.

In the remainder of the paper, Section 2 provides background information on DNS, DNS tunneling, and MSF. Section 3 discusses the different stages of our attack. Section 4 reports statistics on the traffic generated with our attack. Finally, Section 5 draws conclusions.

2 Background and Related Work

2.1 Domain Name System Tunneling

DNS servers store information about computer domain names and their addresses in so-called zone files. Type A records in those files specify (32-bit IPv4) address and name translations such as the fact that www.icisc.org corresponds to IPv4 address 164.125.70.63. AAAA records do the same for 128-bit IPv6 addresses. Canonical Name or CNAME records specify aliases, such as the icisc.org

alias of www.icisc.org. MX or Mail eXchange records specify which servers to use as mail servers and what their priorities are. TXT records can specify a wide range of properties in the form of strings. Typically, these are used to specify constraints on the behavior of mail servers in the Sender Policy Framework [24]. Finally, NS records specify for which domain a DNS server is authorized.

Clients can issue lookup requests for any type of these records with one DNS packet, which is typically transmitted via the UDP protocol [18]. The response to a lookup can only consist of a packet with a record of the same type.

Besides header information, CNAME and TXT packets contain reasonably long strings corresponding to domain names. The other types of packets contain mainly short IP addresses or are so exotic that using them would be too suspicious. So only CNAME and TXT lookups and responses can be used to achieve a reasonable communication bandwidth through a DNS tunnel. For example, some piece of malware installed on a client computer can leak the password `qwerty123` to the malicious `nameserver.evildomain.com` DNS server by sending a TXT lookup for `passwd.qwerty123.evildomain.com`. Being the authoritative DNS server for `evildomain.com`, this DNS server is the last server in the iterative DNS lookup process to receive the lookup. Instead of treating the lookup as a real DNS lookup, this server extracts the communicated information `passwd.qwerty123` from the query. He then sends back information, such as a reboot command embedded in the TXT response `v=spf1 mx a:reboot.1pm.evildomain.com -all`, of which the legitimate meaning is that only local mail servers and the external host `reboot.1pm.evildomain.com` are allowed to send email from senders with email addresses from that same domain, such as `m.romney@evildomain.com`.

To use DNS servers and the DNS protocol as a covert, stealthy communication tunnel, the software implementing the tunnel should exhibit similar behavior as regular DNS traffic. Over ten periods of time, we recorded 10x500 MB of DNS traffic data on our department's local DNS server, which serves our administration and a wide range of research labs accros multiple campuses. We observed that in those periods, TXT records constituted 1–2% of all traffic, CNAME records constituted 20–30%, and A records constitute 38–48%. Furthermore, around 25% of the traffic constituted AAAA records (most of which in support of the Kerberos authentication protocol), and the remaining 5% was spent on NS records. This is in line with other experiments [25].

This implies that a stealthy DNS tunnel can use some TXT records, which can embed longer strings, but that it should mainly rely on CNAME records. This also implies that most if not all information communicated through a DNS tunnel must be encoded in the form of acceptable domain names. We will discuss the practical implications in Section 3.

2.2 Prior Implementations

DNS Cat by Ron Bowes consists of a server and a client application [6]. The client application needs to be invoked explicitly by a user with the domain name of the malicious DNS server (on which the server application runs). The client application is then attached to another process such as a shell, which is from

then on controlled through the tunnel instead of locally. It obtains its standard input from the attacker at the server side through CNAME and TXT response packets sent through the DNS tunnel, and at the same time its standard output is redirected through the tunnel using similar corresponding lookup packets.

DNS Cat can be used in practice to communicate through a DNS tunnel. For example, in hotels with payed internet access, which typically do not filter or block DNS traffic, customers can get online through DNS Cat. The customer can do so because he has full control over a local computer such as his laptop on which he can launch DNS Cat with the appropriate settings. In other words, the customer is a malicious insider. Without help from insiders, however, attacking, e.g., company networks with DNS Cat is not possible.

tcp-over-dns also requires users to start a server-side and a client-side application [2]. Unlike DNS Cat, however, tcp-over-dns enables tunneling any TCP/IP connection through its DNS tunnel. This eases the tunneling of, e.g., browser sessions involving HTTP packets. tcp-over-dns is hence more user-friendly for hotel customers that want (to steal) free Internet access. It is, however, still not useful for attacking company networks without the help from insiders. Iodine [12] (formerly NSTX) and OzymanDNS [11] suffer from the same limitation.

Furthermore, all of these implementations run on the client side as a separate process, which makes them visible to any user or antivirus software.

2.3 DNS Anomaly Detection

Several techniques have been proposed in the past to detect anomalies in DNS traffic, including tunnels and symptoms of other computer network infections.

Some work focuses on relating DNS traffic to real-world events, such as the Tiananmen Square protests [27] or network defects [22]. Detection of network scanning worms has been based on their relatively low number of DNS requests [4,28]. This is obviously not applicable for DNS tunnels, which will operate precisely by executing numerous DNS requests. Fast-flux is a popular and relatively new cyber-criminal technique to hide and protect their critical systems behind an ever-changing network of compromised hosts acting as proxies. The ICANN Security and Stability Advisory Committee gives a clear explanation of the technique [10]. Jose Nazario and Thorsten Holz did some interesting measurements on known fast-flux domains [17]. Fast-flux techniques are mainly orthogonal to the actual launching of (DNS) tunnels and communication through them. The approach by Villamarin-Salomon and Brustoloni focuses on abnormally high or temporally concentrated query rates of dynamic DNS queries [26]. This does not suffice, however, since such patterns also occur for legitimate purposes. Choi et al. check for multiple botnet characteristics based on Dynamic DNS, fixed group activity and a mechanism for detecting migrating C&C servers [7]. They claim that their method works, but the computational demands and processing time seem to prevent it from scaling to large networks. Born and Gustafson propose to use character frequency and N-gram analysis for detecting covert channels in DNS traffic [5]. Their method detects patterns that do not occur in natural languages and are therefore considered anomalous. Master students van

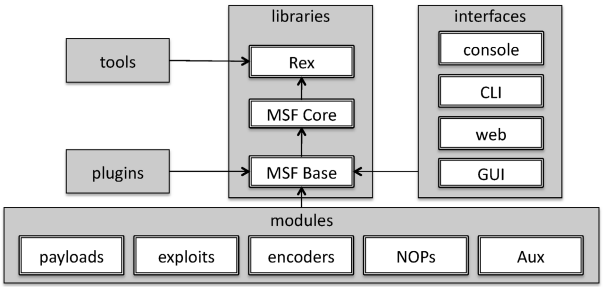

Fig. 1. Metasploit architecture

de Heide and Barendregt provide a preliminary evaluation of the aforementioned techniques without, however, measuring actual implementations [25].

To the best of our knowledge, no existing literature discusses how difficult it is to set up a DNS tunnel without the help of insiders.

2.4 Metasploit

In contrast with existing DNS tunnel implementations, we present an approach that enables outsiders to set up attacks over DNS tunnels without the willing help from insiders. We implemented this approach in MSF [19]. MSF is an open-source framework developed for security experts to ease penetration testing. It consists of a range of tools and modules programmed in Ruby, C and assembler that allow different components of attacks to be reused and combined in different ways. Fig. 1 depicts the MSF architecture. Different (server-side) interfaces are available in the form of a console, a command-line interface (CLI), a web-interface, and a GUI. The most important components for our purpose are the code fragments that will be executed on the client side:

Exploits are small code fragments that can exploit vulnerabilities to take over control of an attacked computer. These fragments can be embedded, e.g., in PDF documents to exploit PDF reader vulnerabilities.

Payloads are the code fragments that attackers want to execute once they have obtained a certain level of control over a machine. There are different types of payloads: *Stagers* try to set up a communication channel between the attacker and the victim as part of a bootstrap process, over which they load stages. *Stages* make up the actual C&C software that gets installed and launched by the stager, such as shell scanning accounts or a spam bot.

Encoders can convert the encoding of a payload without changing its functionality. They consist of packers and unpackers that encrypt and decrypt code to thwart antivirus software, but also of simpler transformations such as removing null bytes from code, to avoid that those bytes are interpreted as the end of a string when trying to exploit buffer overflow vulnerabilities.

NOPs are encoders that can increase the size of payloads to requested sizes by inserting no-operation instructions that do not change the behavior of the payload. This is useful for buffer overflow attacks on fixed sized buffers.

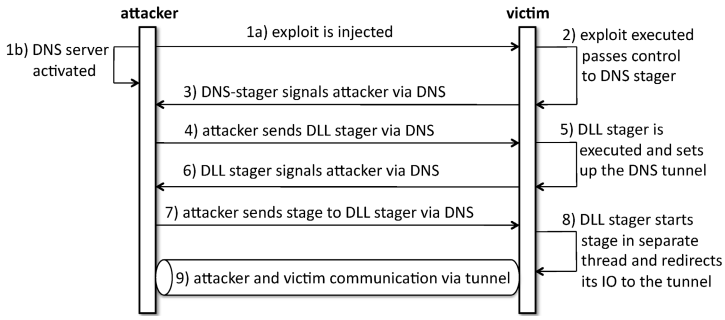

Fig. 2. Bootstrap procedure of our attack

In March 2012, MSF support was added to transmit stages through a DNS tunnel with TXT records [3]. However, that implementation cannot bootstrap from existing, small exploits and does not provide a fully functional, bidirectional DNS tunnel to the stage for later communication of commands and data.

3 Our Staged Attack

This section presents a generic, two-stage DNS-tunneling attack that attackers or security researchers can set up by means of existing exploits of software vulnerabilities, such as buffer overflows, and that can serve as a hidden communication channel for any type of C&C malware. The first stager is very small to allow us to combine it with as many as possible existing MSF exploits. It installs a second stager which is much larger. This stager install a generic DNS tunnel and offers an interface for existing MSF C&C stages to the tunnel. Moreover, the software implementing the tunnel and the interface remain resident (but hidden!) on the attacked computer even when the originally exploited software, such as a PDF reader, is terminated. The source code of our MSF components is available at `https://github.com/azerton/metasploit-framework`. As for integration into MSF, it still needs polishing with respect to code guidelines.

As is common for local MSF exploits, we assume that we can inject a small exploit of a software vulnerability into the network under attack. This injection can happen through mail attachments, web sites infected with drive-by malware, phishing web sites, PDF documents on USB sticks, etc. Injecting this code fragment is orthogonal to the remainder of the attack, and is out of scope of this paper. In the remainder of this section, we discuss the bootstrap process that follows the code injection, as depicted in Fig. 2. We focus on the software we developed to run the computer under attack, i.e., the stagers, as the server side consists mostly of standard MSF functionality adapted slightly for our purpose.

3.1 Stage 1: DNS Stager

The DNS stager is the piece of code that will be injected in a process on the victim computer together with the exploit to hijack that process. The most

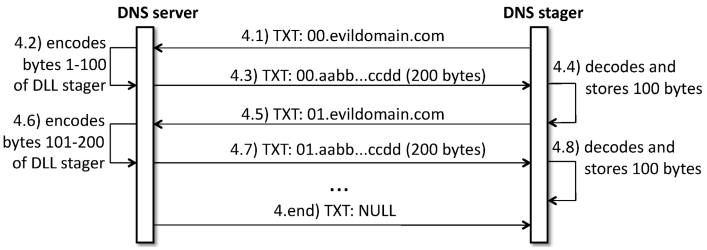

Fig. 3. Step 4: Downloading of the DLL stager by the DNS stager

commonly used technique to inject code is by means of buffer overflows. At the time of writing, about 68% of all MSF exploits rely on buffer overflows.

Targeting buffer overflows constraints the stager. First, many targeted buffers are small. Hence the stager has to be made as small as possible to fit as many as possible buffer overflow exploits. Secondly, many buffer manipulation functions in software treat certain characters in a special way. For example, string copying ends on a null byte, and FTP-servers treat ampersands and newline characters differently. So some characters should preferably not occur in the stage. Finally, the stager's code has to be as platform-independent as possible, and directly executable at any address at which it is injected. In our case, we opted for position-independent x86 assembly code (PIC) that targets Windows systems.

Starting from Ron Bowes' DNS Cat stager implementation that was only suitable to stage short shell scripts encoded as ASCII text [6], we have written a stager consisting of 168 x86 assembler instructions that occupy 518 bytes. This stager is capable of coordinating steps 3) and 4) in Fig. 2.

Some additional features our stager supports are the transmission of binary data (e.g., the DLL stager's code) through TXT packets by means of NetBIOS coding and decoding, supports for ten times more TXT packets than DNS Cat to enable the transmission of the bigger DLL stager, and allocation of much more heap memory for storing large MSF payloads.

The actual transmission of the DLL stager under coordination of the DNS stager is depicted in Fig. 3. All transmitted packets are numbered to compensate that the UDP protocol does not guarantee delivery, ordering or duplicate protection. While CNAME records are stealthier as discussed above, we use TXT packets for this first stage of the attack because parsing TXT records is easier, and hence can be done with less code than parsing CNAME records. We have chosen to limit the packet data length to 200 bytes in our implementation, but this can easily be adapted. This is much shorter than the theoretical upper limit of 64K bytes per TXT packet because we observed that many software implementations in DNS servers cannot handle atypically long packets, and because IDPSs might consider atypically long DNS packets as suspicious. Switching to even shorter packets will increase the number of packets and hence the amount of time needed to transfer and install the DLL stager. This can be problematic for exploits that result in the process they hijacked being killed. If the process is killed before the DLL stager is up and running, the attack will fail.

Finally, we should note that the NetBIOS [14] coding we use is not the most efficient way to encode binary data in ASCII strings with respect to communication bandwidth. It converts each 4-bit nibble into an ASCII character by adding 0x41 to it. So it uses only 16 of the more than 26 lowercase + 26 uppercase + 10 digits + punctuation possible values. There are two reasons for using this encoding. NetBIOS decoding is simple enough to be supported in the very small stager of 168 instructions. Secondly, domain names are case-insensitive. For efficiency reasons, e.g., to save space in caches, DNS server software typically use normalized lower-case names. Hence upper-case characters risk not surviving the passage through DNS servers not controlled by the attacker.

3.2 Stage 2: DLL Stager

The DLL stager differs significantly from the DNS stager with respect to the way in which it is programmed, launched and executed. Whereas the first, DNS stager was programmed in PIC assembly to be injected and executed directly in the hijacked process, the second stager is too complex to be programmed in assembler. And whereas the DNS stager only has to provide a tunnel for itself, i.e., a tunnel that only supports loading the DLL stager, the DLL stager has to set up a more generic tunnel for use by the stages, keep that tunnel alive, and offer the stages an interface to it. For the latter two reasons, the DLL server needs to stay alive even after the originally hijacked process is killed. But it should stay alive in a way that is not easily detected, and hence not in a new, separate process. In other words, the DLL stager has to be injected into and operate in another, longer running process on the victim computer. The most convenient way to inject code into a running process and execute it, is by means of reflective DLLs [9,23]. Reflective DLL injection is a library injection technique in which the concept of reflective programming is employed to perform the loading of a library from memory into a host process. As such the library is responsible for loading itself by implementing a minimal Portable Executable (PE) file loader [13]. We implemented the second stager as such a reflective DLL, hence the name DLL stager.

When the 168 instruction DNS stager has downloaded the DLL stager via DNS, this DLL stager is launched (step 5 in Fig. 2). This stager is responsible for downloading one or more MSF stages as shown in Fig. 4. It will do so in a very similar way as the DNS stager, but there are some significant differences. First, CNAME packets will be used instead of TXT records to avoid detection. This implies that also in the responses, a considerable amount of space is spent on repeating domain names. Secondly, besides order numbers, the communicated domain names also includes session numbers, which allows an attacker to set up multiple concurrent sessions from within the same network. Furthermore, both the DNS server and the DLL stager add a third random number between 0 and 100 to the transmitted domain names to prevent the DNS servers from caching requests [6]. This enables the DLL stager to load multiple different stages if wanted. What remains of the 63 characters that are available in a CNAME packet is filled with useful information, such as the `GETPAYLOAD` and the NetBIOS string

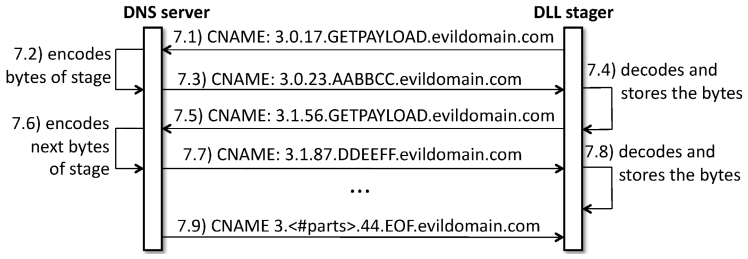

Fig. 4. Step 7: Downloading of the stage by the DLL stager

`AABBCC` that represents stage code in Fig. 4. On the server side, MSF encoders automatically ensure that the transmitted stages (which over time may evolve independently from the DLL stager in the MSF) are in the least vulnerable format as chosen by and from the perspective of the attacker. This facilitates the integration and maintenance cost of our DNS and DLL stagers in the MSF.

When the DLL stager has downloaded a stage (step 7 in Fig. 2), it injects it as a new thread in the same application in which the DLL stager was injected itself, as chosen by the DNS stager. This injection is step 8 in Fig. 2. From that point on, the DLL stager becomes the bridge between the stage and the DNS tunnel. Internally, that phase of the DLL stager is designed as two cooperating components, as depicted in Fig. 5.

The DNS tunnel client is responsible for setting up and keeping alive the DNS tunnel. It implements the encoding and decoding of all information transmitted and received through CNAME packets, as was done for loading the stage(s). The component uses session numbers and packet order numbers to overcome reliability issues of the UDP protocol and to support multiple sessions. Furthermore, this component implements a form of polling through the DNS tunnel. In the DNS protocol, DNS servers can only respond to queries from clients. This means that a malicious DNS server cannot initiate any communication with a C&C client. Many such clients are designed to wait for commands from a server, however, without explicitly asking for such commands. In other words, stages in Fig. 5 might simply be sleeping and waiting to be woken up by a command. To allow the server to send commands that wake up sleeping clients, the DNS tunnel client sends lookups to the server of an initiated session on a regular basis. Whenever the server wants to send a command, it does so in a response to the polling lookup, which the DLL stager then forwards to the stage.

To facilitate the communication between client stages and servers without having to adapt the stages to the fact that they use a DNS tunnel, a second DLL stager component offers a TCP abstraction of the tunnel client. As a result, stages only have to communicate with socket pairs, which is a fairly standard method. To implement this component without having to implement all base functionality ourselves, we relied on the standard `winsock2` library.

As a whole, this stager consists of two parts: the reflective loader present only to install the stager, and the stager itself. Like the DNS stager, the reflective loader consist of manually engineered PIC (in this case written in heavily

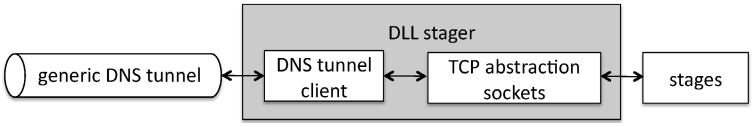

Fig. 5. Components of the DLL stager

constrained C code). Whenever this code wants to perform an API call, it first computes the address of the callee itself. These features are necessary because the reflective loader is invoked without its binary code getting relocated by the standard OS loader [13]. By avoiding the use of the standard OS loader, we also avoid the need to embed full PE headers in the binary and it allows us to put the DLL on the standard heap instead of on separate pages. Avoiding the standard loader, full headers and separate pages make the DLL stager much stealthier for antivirus software, which typically attaches itself to the standard loader to monitor the binaries being loaded.

In a first attempt, we implemented the second part, i.e., the DLL stager components, in C++. While this was very productive from a software-engineering perspective, the compiled DLL stager proved to be too big. This posed no technical problems, but it did increase the number of TXT packets that had to be transmitted to download the DLL stager, which increases the change of being detected. We therefore reimplemented the DLL stager in C, which resulted in an acceptable total binary size of 61KB.

4 Evaluation

To evaluate our implementation, we have set up our own subdomain DNS server with the free `www.afraid.org` service for static and dynamic DNS domain and subdomain hosting. This DNS server was the authoritative server for the azertontunnel.chickenkiller.com domain, a relatively long name that has to be included in each DNS lookup. We use the Google DNS server at 8.8.8.8 as a primary DNS server. This service is free and requires no authentication, which is perfect for an automated attack. In an alternative experiment, we installed a server and client locally, such that all information is sent directly from victim to server and back, without being delayed by external DNS severs. We refer to the first and second experiments as the global and local experiments respectively. In both experiments, the stage consisted of a small shell that runs C&C commands to obtain system information, of which the output is transmitted to the attacker through the DNS tunnel.

4.1 Throughput

First, we measured the maximal throughput we obtained with the described staged attacks, i.e., the amount of useful data an attacker can transmit. This excludes, e.g., the overhead bytes to embed the purposely long domain name `azertontunnel.chickenkiller.com` in the packets and the packet headers. To

Table 1. Observed throughputs

phase	DNS stager	DLL stager	shell
record type	TXT	CNAME	CNAME
data data size	DLL stager 61KB	shell 0.23KB	system info & polling 40KB
local throughput	8.14 KB/s	0.80 KB/s	3.34 KB/s
global throughput	0.68 KB/s	0.68 KB/s	2.18 KB/s

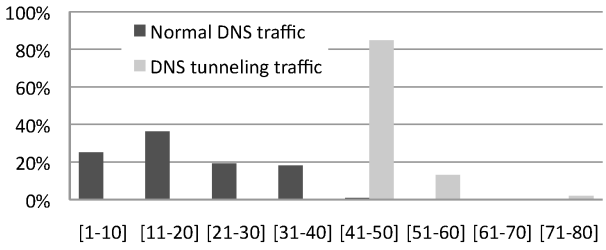

Fig. 6. Histogram of CNAME packet lengths

measure the throughout of the C&C shell, we ran an automated script on the attacker's side. In the evaluated implementation, the TXT records were limited to 200 bytes of useful information per DNS packet, and the CNAME records to 16 bytes, which we estimated to result in non-suspicious packets. Table 1 presents the throughput results.

The most important observation is that TXT records used by the DNS stager proved to give higher throughput only on the local network. It is unclear why the DNS stager in the global experiment is slowed down to the same level as the DLL stager. We suspect that it has to do with prioritization in DNS servers. As delays in A records and in CNAME records are typically more noticeable to clients, DNS servers might be handling those with higher priority. Further research is needed to clarify this. Even when we don't understand the cause of this behavior completely, we can conclude from this experiment that for global attacks, the DNS stager might as well use CNAME records. This will not slow the attack down, while at the same time making the attack more stealthy. As discussed in Section 3, it does increase the risk of attacks not succeeding, however.

Furthermore, we observe a shell C&C throughput of 2.18 KB/s, which is plenty for many real-world attacks, such as for stealing passwords, credit card numbers, and PINs by means of keyloggers.

In our experiments, the stage itself obtains a higher throughput than the DLL stager obtains while loading the stage. The reason is that the DLL stager's code that handles the stage's data handles this data more efficiently than the code in the DLL stager that downloads the stage itself.

4.2 Packet Sizes

With interactive C&C sessions, rather than automated attack scripts, we measured the DNS packet sizes to evaluate their stealthiness. The packet sizes,

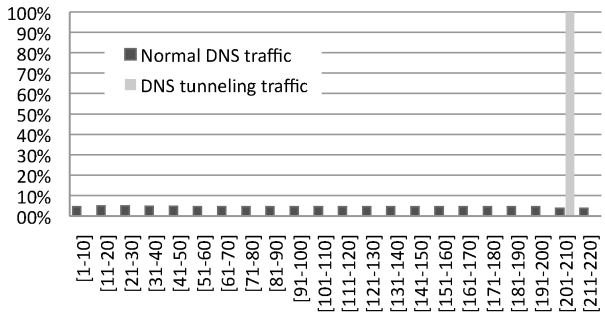

Fig. 7. Histogram of TXT packet lengths

including the overhead of domain names but not the fixed size headers, are depicted in Figs. 6 and 7.

It is clear that even with a limitation of 200 and 16 bytes per TXT and CNAME record, the packet lengths of tunneled traffic are easily distinguished from normal traffic. So IDPS could, in theory, easily detect and block our tunnel.

To prevent this, an attacker can in practice rely on CNAME packets only, which will in practice not lower his throughput as discussed above, limit the number of useful bytes per packet and use a shorter domain name than our purposely long `azertontunnel.chickenkiller.com`. He might then reach a lower throughput than 2.18 KB/s, but it will still be enough to obtain the most valuable, privacy-sensitive information.

5 Conclusions and Future Work

With the presented proof-of-concept Metasploit prototype, we have demonstrated that it is possible to set up fully functional DNS tunnels to private networks starting from small local exploits such as buffer overflows, i.e., without the willing help from insiders, and to use those tunnels for command-and-control attacks. This provides a strong incentive for firewalls and intrusion detection systems to start monitoring the often neglected DNS traffic. Our current implementation is probably not stealthy enough to avoid detection by adapted protection systems, but there seems to be plenty of room (i.e., bandwidth) to make it stealthier, so more research in this direction is needed in the future.

References

1. Amon, C., Shinder, T.W., Carasik-Henmi, A.: The Best Damn Firewall Book Period, 2nd edn. Syngress Publishing (2007)
2. AnalogBit: tcp-over-dns, `http://analogbit.com/software/tcp-over-dns`
3. Beardsley, T.: Weekly Metasploit Update: DNS payloads, Exploit-DB, and More. Rapid7 Blog Post (March 2012), `https://community.rapid7.com/community/metasploit/blog/2012/03/28/metasploit-update`

4. Binsalleeh, H., Youssef, A.: An implementation for a worm detection and mitigation system. In: Proc. 24th Biennial Symposium on Communications, pp. 54–57 (June 2008)
5. Born, K., Gustafson, D.: Detecting DNS tunnels using character frequency analysis. In: Proceedings of the 9th Annual Security Conference (April 2010)
6. Bowes, R.: DNS Cat, http://www.skullsecurity.org/wiki/index.php/Dnscat
7. Choi, H., Lee, H., Lee, H., Kim, H.: Botnet detection by monitoring group activities in DNS traffic. In: Proc. 7th IEEE Int. Conf. on Computer and Information Technology, pp. 715–720 (2007)
8. Di Pietro, R., Mancini, L.V.: Intrusion Detection Systems, 1st edn. Springer Publishing Company, Incorporated (2008)
9. Fewer, S.: Reflective DLL injection. Technical Report, Harmony Security (2008)
10. ICANN Security and Stability Advisory Committee: SSAC advisory on fast flux hosting and DNS (2008)
11. Kaminsky, D.: OzymanDNS, http://en.cship.org/wiki/OzymanDNS
12. Kryo: iodine, http://code.kryo.se/iodine
13. Levine, J.: Linkers & Loaders. Morgan Kaufmann Publishers (2000)
14. Microsoft Corporation: ASCII and hex representation of NetBIOS names, http://support.microsoft.com/kb/194203
15. Mockapetris, P.: RFC 1034 Domain Names - Concepts and Facilities. The Internet Engineering Task Force, Network Working Group (November 1987)
16. Mockapetris, P.: RFC 1035 Domain Names - Implementation and Specification. The Internet Engineering Task Force, Network Working Group (November 1987)
17. Nazario, J., Holz, T.: As the net churns: Fast-flux botnet observations. In: Proc. 3rd International Conference on Malicious and Unwanted Software, pp. 24–31 (October 2008)
18. Postel, J.: RFC 768 User Datagram Protocol. The Internet Engineering Task Force (August 1980)
19. Rapid7: Metasploit framework, http://www.metasploit.com
20. Rapid7: Metasploit pro user guide, http://community.rapid7.com/docs/DOC-1501
21. Rebane, J.C.: The Stuxnet Computer Worm and Industrial Control System Security. Nova Science Publishers, Inc., Commack (2011)
22. Shin, H.J.: A DNS anomaly detection and analysis system. NANOG 40 (June 2007)
23. "skape", Turkulainen, J.: Remote library injection. Technical Report, nologin (2004)
24. The SPF Council: Sender policy framework, http://www.openspf.org/
25. van der Heide, H., Barendregt, N.: DNS anomaly detection. Technical Report, Universiteit van Amsterdam (2011)
26. Villamarin-Salomon, R., Brustoloni, J.: Identifying botnets using anomaly detection techniques applied to DNS traffic. In: Proc. 5th IEEE Consumer Communications and Networking Conference, pp. 476–481 (January 2008)
27. Whang, Z., Tseng, S.S.: Anomaly detection of domain name system (DNS) query traffic at top level domain servers. Scientific Research and Essays 6(18), 3858–3872 (2011)
28. Whyte, D., Kranakis, E., van Oorschot, P.: DNS-based detection of scanning worms in an enterprise network. In: Proc. of the 12th Annual Network and Distributed System Security Symposium, pp. 181–195 (2005)