Chapter 1

AN FPGA-BASED SYSTEM FOR DETECTING MALICIOUS DNS NETWORK TRAFFIC

Brennon D. Thomas, Barry E. Mullins, Gilbert L. Peterson and Robert F. Mills

Abstract Billions of packets traverse computer networks every day. Often, these packets have legitimate destinations such as buying a book at amazon.com or streaming a video. Unfortunately, malicious and suspicious network traffic continues to plague the Internet. One example is abusing the Domain Name System (DNS) protocol to exfiltrate sensitive data, establish backdoor tunnels, or control botnets. To counter this abuse and provide better incident detection, a forensic tool is developed to detect suspicious DNS traffic. The forensic tool is derived from the TRacking and Analysis for Peer-to-Peer (TRAPP) system, developed in 2008, to detect BitTorrent and Voice over Internet Protocol (VoIP) traffic of interest. Using concepts and technology developed for the TRAPP system, the new TRAPP-2 system forensic tool is constructed on a Xilinx Virtex-5 ML510 Field Programmable Gate Array (FPGA) board. The TRAPP-2 system detects a DNS packet, extracts the payload, compares the data against a hash list and, if the packet is suspicious, logs the entire packet for future analysis. The goal of this research is to evaluate the performance of the TRAPP-2 system as a solution to detect and track malicious DNS packets traversing a gigabit Ethernet network. Results show that the TRAPP-2 system captures 91.89% of DNS packets of interest while under a 93.7% network load (937 Mbps). In another experiment, the hash list size is increased from 1,000 to 131,072,000 unique items and reveals that each doubling of the hash list size results in a mean increase of approximately 16 central processing unit cycles. These results demonstrate the TRAPP-2 system's ability to detect traffic of interest under a saturated network while maintaining large hash lists.

Keywords: FPGA, DNS, tunnel, network forensics tool, exfiltration, gigabit

1. Introduction

Malicious and suspicious network traffic continues to plague the Internet. Recent incidents include blueprints for Marine One being leaked by a United States contractor using a file sharing program [1], and Chinese hackers pilfering intellectual property from Google and other United States companies [2].

As a result of these growing threats, the TRacking and Analysis for Peer-to-Peer (TRAPP) system was developed in 2008 to detect two protocols possibly transferring malicious content [3]. The first is the BitTorrent protocol used to transfer and share files, and the second is the Session Initiation Protocol (SIP), used to setup and close Voice over Internet Protocol (VoIP) calls. The system resides on a Xilinx Virtex-II Pro Field Programmable Gate Array (FPGA) board. This first iteration prototype is limited in its 100 megabit Ethernet controller, small hash list size, and lack of malicious Domain Name System (DNS) network traffic detection. However, TRAPP still captures packets of interest with a "probability of intercept of at least 99.0%, using a 95% confidence interval and given an 89.6 Mbps network utilization" [3]. These results show the TRAPP system is a viable network forensic tool worth expanding to incorporate a gigabit Ethernet controller, larger hash list sizes, and malicious DNS detection.

This research extends the first TRAPP system [3] by implementing a more powerful FPGA board, but only focuses on the DNS protocol. The research consists of creating a second generation TRAPP system, named TRAPP-2, designed on a Xilinx Virtex-5 ML510 FPGA board with a faster processor and a gigabit Ethernet controller [4]. The FPGA board is chosen because of the speed, cost, and ease of both hardware and software configuration. Ultimately, the research determines that the TRAPP-2 system is a feasible solution to detect and track malicious DNS requests on gigabit Ethernet networks. The TRAPP-2 system meets two measurement goals. The first goal determines the probability of intercepting a malicious DNS packet under various network loads. The second goal determines how increasing the hash list size affects the packet processing time. The two metrics used to measure performance are probability of packet intercept and packet processing time measured in Central Processing Unit (CPU) cycles.

2. Background and Related Work

2.1 Domain Name System Tunneling

DNS is a critical service for the Internet. However, the DNS protocol can be exploited for nefarious purposes. One method of abusing the protocol is DNS tunneling, as first suggested in a 1998 Bugtraq posting by Oskar Pearson [5]. DNS tunneling is an abuse of DNS records to transfer non-DNS data in and out of a network using the DNS protocol. Non-DNS data can include files, botnet commands, and even segmented audio media [6]. DNS tunneling is appealing because it is a covert channel and is operating system-independent [6].

The concept of the DNS tunnel is to use a hacker-controlled DNS server as an external trusted server to tunnel information out of a protected network through standard DNS traffic. The assumptions are that a hacker has already compromised the victim's computer, installed a DNS tunneling program, and is not using Secure Shell along with SOCKS 4/5 to tunnel DNS queries. Since most protected networks permit DNS traffic to exit, this "infected" DNS traffic is allowed unabated. The data are transmitted through the tunnel by sending data to the hacker's DNS server in the form of a query and getting data back in the form of a response [6]. Typically, the tunneled data appears as the DNS request [exfiltrated data].hacker.com, with the data residing in the lowest level domain.

Figure 1 summarizes the process in five steps. The victim's computer performs a DNS request for [exfiltrated data].hacker.com. The DNS request for [exfiltrated data].hacker.com is not locally cached so it requires the Company X DNS server to resolve the request. Company X's DNS server cannot resolve the request, so it forwards it to the DNS server under the hacker's control at hacker.com. The hacker sends back a DNS response which easily passes through a network defense appliance since DNS is assumed to be trusted. The victim receives the DNS response to exfiltrate more data, connect to a botnet, etc.

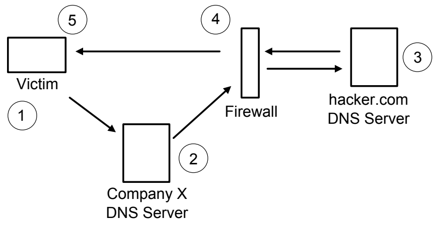

Figure 1. Establishing a Domain Name System Tunnel.

Some DNS tunneling applications include Iodine [7], OzymanDNS [8], NSTX (Nameserver Transfer) [9], and Heyoka [10]. For this research, Iodine is selected to create a DNS tunnel, and Wireshark is used to capture the tunneled DNS packets used for the experiment. The Iodine DNS packets are created and captured outside the experiment. Iodine is capable of tunneling IP traffic through DNS and offers more benefits over other DNS tunnel implementations such as portability between systems, an MD5 challenge-response for login, and the use of the NULL DNS record type to allow unencoded downstream data which allows up to a kilobyte of compressed payload data [7].

2.2 Illicit Traffic Detection

Methods currently available to detect illicit DNS traffic include software-based solutions through utilizing artificial intelligence. Software-based solutions include Wireshark [13], Snort [14], and the Hi-Performance Protocol Identification Engine (HiPPIE) [15]. All three software-based solutions require more processing time since they operate at the application layer. Malicious DNS traffic can be detected using entropy-based systems. Romana *et al.* performed an entropy study of external DNS query traffic to the university's top domain server. Any peak in the entropy was assumed, and later proven, to be associated with spam botnet activity [16]. Karasaridis *et al.* developed a DNS Tunneling Attack Detector (TUNAD) to detect suspicious DNS packet size anomalies in real-time [17]. The disadvantage of the entropy-based solutions is that they lack real-time detection. Lastly, jhind utilized artificial neural networks to measure the entropy of previously captured tcpdump files, thus also lacking real-time detection [18].

2.3 The TRAPP System

An FPGA-based packet analyzer was developed in 2008 to detect peer-to-peer protocols transferring malicious content across a network. The TRAPP system was built specifically to detect BitTorrent and Session Initiation Protocol (SIP) packets. It was created as an alternative to current illegal network traffic detection methods and is designed to operate at the gateway between the Internet and a local area network (LAN). It is not placed in-line with traffic entering or exiting the local network; therefore, if the TRAPP system fails, the network will remain unaffected. Specifically, it is placed on the Switched Port Analyzer (SPAN) port of a switch. This makes the TRAPP system virtually undetectable to normal and malicious users.

The TRAPP system analyzes every packet flowing through the network switch in real-time, looking for a BitTorrent or SIP signature. If the packet has a BitTorrent or SIP signature, the TRAPP system extracts the first 32 bits of the BitTorrent file hash or the first 12 bytes of the SIP Uniform Resource Identifier (URI) and compares it against a list of known contraband file hashes or SIP identifiers. A binary search is performed on the extracted hash against a blacklist of BitTorrent hashes or SIP URIs. If a match is found, the packet is logged; otherwise, TRAPP ignores the packet [3].

There are several limitations with the TRAPP system including the hardware, contraband file hash list size, and lack of malicious DNS traffic detection [3]. The hardware components of interest on the Xilinx Virtex-II Pro FPGA board are the 100 megabit Ethernet controller and 300 MHz processor [19], which are suitable for smaller LANs. In reality, the size of modern networks, traffic, and bandwidth requirements justify faster hardware. Another drawback of the TRAPP system is the size of the contraband hash list. The TRAPP system relies on 64 KB of memory to store the contraband hash list, thus limiting it to approximately 16,000 entries [3]. This size is appropriate for a first iteration proof-of-concept system, but in reality, the list size needs to be much larger. Lastly, the TRAPP system is unable to detect illicit DNS traffic [3].

3. TRAPP-2 Methodology

The TRAPP-2 system detects DNS packets of interest traversing a gigabit Ethernet network. The TRAPP-2 system detects these transmissions, classifies the traffic, extracts the payload, performs a hash on the DNS domain, compares the hash against a whitelist, and records the transmission information.

3.1 Hashing Function

A new feature of the TRAPP-2 system is the implementation of a hashing function used in the Substitute Database Manager (sdbm) library [11]. The hashing function converts arbitrary-length strings into four-byte hashes. The arbitrary-length strings in this case are DNS domains. The sdbm hashing function is selected over more proven hashing functions, such as Secure Hash Algorithm 1 (SHA-1) and Message Digest algorithm-5 (MD5), because it is quick and easy to implement [12]. The hashing function's speed, four-byte hashes, and easy software implementation make it an ideal hashing function for the TRAPP-2 system. While using the hashing function, one drawback occasionally noted is the minimal avalanche effect in which changing a DNS domain by one bit (e.g., from 122.com to 123.com) changes the hash by one bit. Another possible drawback is the number of collisions between hashes. This, however, is not investigated in the research.

3.2 System Algorithm

Figure 2 illustrates the TRAPP-2 algorithm. For DNS packets, the system detects a DNS request, extracts the entire domain, sdbm hashes the domain to create a four-byte unique hash, compares the hash against a whitelist of approved domain hashes, and logs it if it is not on the DNS whitelist. A DNS request is defined as a UDP packet with a destination port of 53. DNS zone transfers, performed over TCP port 53, are not included because they are not capable of exfiltrating data.

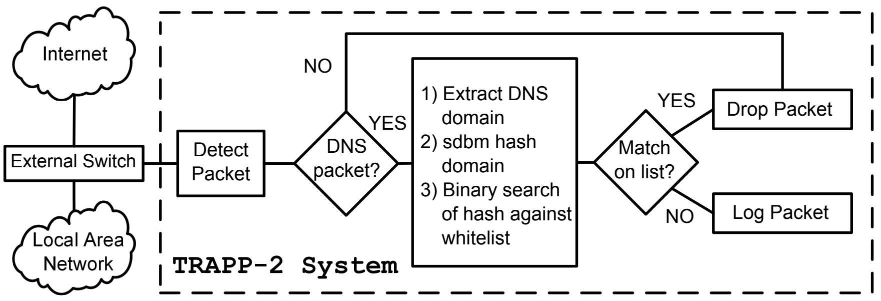

Figure 2. Packet Data Flow in the TRacking and Analysis for Peer-to-Peer 2 System.

3.3 Evaluation Technique

Direct measurement is selected as the evaluation technique for the experiments because the TRAPP-2 system is a real and physical system. The experimental hardware configuration setup in Figure 3 consists of the following hardware:

- 1 Cisco gigabit 24-port switch (model WS-C3560G-24PS-S). The switch is configured with 22 standard ports and 2 SPAN ports.
- 1 Xilinx Virtex-5 FPGA board (model FXT ML510) connected to one of the switch's SPAN ports.
- 1 Dell Latitude D630 laptop loaded with the Windows XP Service Pack 3 Operating System. It contains Wireshark 1.0.5 [13], connected to the other switch's SPAN port, acting as the control packet sniffer. This laptop is also used to program the FPGA via Universal Serial Bus and provide Standard Input/Output for the FPGA board through a RS232 interface.

- 1 Dell Latitude D630 laptop loaded with Backtrack 4 [20] and tcpreplay, version 3.4.3 [21], to inject packets into the network.
- 1 Dell Latitude D630 loaded with the Ubuntu Desktop 9.10 Operating System. This laptop contains the Linux pktgen utility to create different network loads on the network.

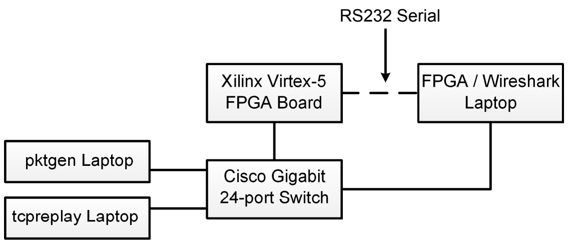

Figure 3. Experimental Hardware Configuration Setup for the TRacking and Analysis for Peer-to-Peer 2 System.

3.4 Performance Metrics

The two performance metrics used to evaluate the effectiveness of the TRAPP-2 system are probability of packet intercept and packet processing time. Network load is also discussed because it is measured in Experiment 1.

Probability of packet intercept is calculated by determining if a packet of interest is captured and successfully recorded to the log file. When measuring the probability of packet intercept, the network load of the system is also measured. Experiment 1 measures the probability of packet intercept of a DNS packet of interest while adding a non-DNS traffic load to the TRAPP-2 system.

For Experiment 1, 300 packets are sent at 200 ms intervals from the Backtrack laptop using tcpreplay. Injecting the packets at 200 ms intervals allows for the result of each trial (captured or not captured) to be independent. The sample size of 300 packets produces a good binomial distribution with small confidence intervals. For each of the three replications, 300 packets are sent into the TRAPP-2 system and the number of packets captured is recorded. Prior to sending the 300 packets, five packets are sent to the system to "warm up" the board by caching the data and instructions used by the processor. Additionally, prior to injecting the 300 packets, pktgen is activated to add the various network loads to the system. Network load is measured using variables within pktgen. For Experiment 1, Wireshark is used as the probability of packet intercept control.

The second metric is the packet processing time, which measures the CPU cycles required to process packets. The PowerPC's System Timer timestamp function is used to tag when a packet arrives in the Ethernet controller and when the packet has been completely processed. For Experiment 2, a series of 50 packets is sent from the Backtrack laptop using tcpreplay, which allows for sufficiently small confidence intervals to compare the results. For each of the three replications, 50 packets are sent and the number of CPU cycles required to process the packet is recorded. Prior to sending the 50 packets, five packets are once again sent to "warm up" the system. No additional network load is injected into the system.

Network load is the total amount of traffic entering the TRAPP-2 system. For Experiment 1, network utilization varies with the load generated by pktgen. For Experiment 2, network load is limited to single packets injected into the system, and is thus virtually zero.

3.5 Approach

The TRAPP-2 system is developed on the Xilinx Virtex-5 FXT ML510 FPGA board. The reason for developing the TRAPP-2 system on an FPGA board is similar to the original TRAPP system, henceforth referred to as "TRAPP-1". The system's simplicity and speed is maximized by allowing the software application to directly access the Ethernet controller buffers [3]. In addition, hardware components and software modifications can easily be added with minimum overhead. Some elements and functions from the TRAPP-1 system are used for the TRAPP-2 system. Although both systems work similarly, major hardware and software modifications are required to achieve proper functionality in the TRAPP-2 system.

The major hardware modification between the TRAPP-1 and TRAPP-2 systems is the Ethernet controller. The TRAPP-1 system relies on the EthernetLite core peripheral, which has an upper limit of 100 Mbps. For the TRAPP-2 system, a Trimode Ethernet Media Access Controller is used to receive Ethernet frames at 1000 Mbps. An accompanying First-In-First-Out 32,768-byte (maximum allowed) buffer stores Ethernet frames until they can be processed.

The second hardware modification is the memory location of the hash list and log file. The hash list contains a sorted list of hashes used to determine if a DNS packet hash is of interest, and the log file contains all of the packets of interest detected by the TRAPP-2 system. The TRAPP-1 system relies on two sets of 64 kilobyte Block Random Access Memory (BRAM) to separately store the hash list and log file. The maximum amount of BRAM available on the TRAPP-2 system's FPGA is 128 kilobytes per block. This severely limits the maximum hash list size, which is explored in Experiment 2. As a result, the BRAM architecture is abandoned in favor of a 512 megabyte Synchronous Dynamic Random Access Memory (SDRAM) scheme to store the hash list and log file together on the TRAPP-2 system. Pilot tests reveal an average increase of 777 CPU cycles to detect and process a DNS packet using the SDRAM scheme. However, the 4096-fold gain in physical memory address space at the cost of 777 CPU cycles is deemed acceptable. This memory configuration is also more realistic for future configurations that will rely on larger hash lists.

The software modification involves the hashing of DNS domains. A uniform hash length is required for proper binary searching of the hash list. However, the variable-length of DNS domains does not allow for a uniform hash list. As a result, the sdbm hash is implemented to convert the variable-length DNS domains into a four-byte hash. Pilot tests reveal that an average of 86 CPU cycles is required to sdbm hash a six character domain name and 1,195 CPU cycles are required to sdbm hash a 212 character domain name. This 86 - 1,195 CPU cycle increase in packet processing time is acceptable.

4. Experiments

Two partial factorial experiments are conducted. Experiment 1 determines the probability of packet intercept under various network loads generated using pktgen. Experiment 2 determines how increasing the hash list size affects packet processing time.

4.1 Workload and System Parameter

The workload for the TRAPP-2 System Under Test consists of a DNS packet and a network load. The malicious DNS packet is created using the DNS tunneling program Iodine prior to the experiments [7].

For Experiment 1, a network load consisting of non-DNS traffic is added to the system using the Linux pktgen utility [22]. By adding the load, the resulting minimum network utilization is approximately 20% (204 Mbps) and is increased at 10% intervals up to the maximum achievable rate of 93.7% (equivalent to 937 Mbps).

The pktgen utility is a Bourne Again SHell (BASH) script that runs in a terminal and allows configuration of the packet size, number of packets, and delay. The number of packets and packet size remain static at 6,000,000 packets and 1,500 bytes, respectively. The delay variable is modified to achieve the different network load percentages. A timestamp function within the BASH scripting language is used to record the number of nanoseconds since January 1, 1970. This timestamp function is taken immediately before pktgen begins and immediately after completion. As a result, the total time required to send the 6,000,000 packets is known, and the megabits per second network load can be calculated.

The single TRAPP-2 system parameter is the hash list size. For Experiment 1, a hash list size of 1,000 is used. For Experiment 2, the hash list size is doubled from 2,000 up to 131,072,000 unique hash items resulting in 17 different hash list sizes. The hash list with 131,072,000 items is 500 MB, which is 97.65% of the available SDRAM. The hash list size is capped at this value to accommodate the log file which shares this memory.

4.2 Experiment 1

Experiment 1 is a partial factorial design that calculates the probability of packet intercept for the DNS packet. Experiment 1 is performed under eight different non-DNS network loads, generated by pktgen, and consists of 7,200 trials (1 packet type x 300 packets x 8 loads x 3 replications). A one-proportion confidence interval analysis is performed on the binomial variable to determine the probability of packet intercept and a 95% confidence interval for the proportion.

Figure 4 shows a plot of the probability of packet intercept for both the TRAPP-2 system and Wireshark as the network load is increased. Confidence intervals are calculated but not shown since they are small and virtually undetectable on the plot.

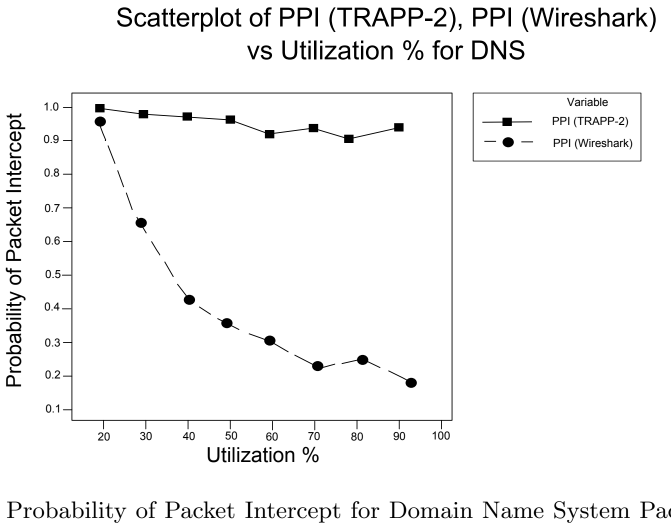

Figure 4. Probability of Packet Intercept for Domain Name System Packets versus Various Network Loads for Experiment 1.

The TRAPP-2 system has a higher probability of packet intercept for every network utilization level. Figure 4 reveals the approximate and slight linear decrease in probability of packet intercept for the TRAPP-2 system, as opposed to the exponential decrease by Wireshark as network utilization increases. This is further emphasized by the fact that the TRAPP-2 system manages to capture 91.89% of DNS packets at the maximum network utilization of 93.7%. In contrast, Wireshark only captures 18.00% of DNS packets at the maximum network utilization. The default buffer size for Wireshark, which is used for this research, is 1 MB. This is significantly greater than the 32 KB First-In-First-Out buffer used in conjunction with the FPGA's Ethernet controller. Perhaps increasing the buffer size in Wireshark can produce more favorable results, but the fact still remains that the TRAPP-2 system's buffer is smaller and outperforms Wireshark.

4.3 Experiment 2

Experiment 2 is a partial factorial design and calculates the packet processing time for a DNS packet. Experiment 2 consists of 2,550 trials (17 list sizes x 1 packet type x 50 packets x 3 replications). A one variable t-test is used to determine the mean packet processing time in CPU cycles, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean.

Figure 5 shows a plot of the mean packet processing time as the hash list size is increased. Once again, confidence intervals are calculated but not shown. The smallest hash list size is 2,000 and is doubled up to 131,072,000 unique hash items on the hash list. The doubling of the hash list size results in a logarithmic plot for the mean packet processing times. The difference between mean packet processing times for the hash list size of 2,000 and 131,072,000 is approximately only 255 CPU cycles.

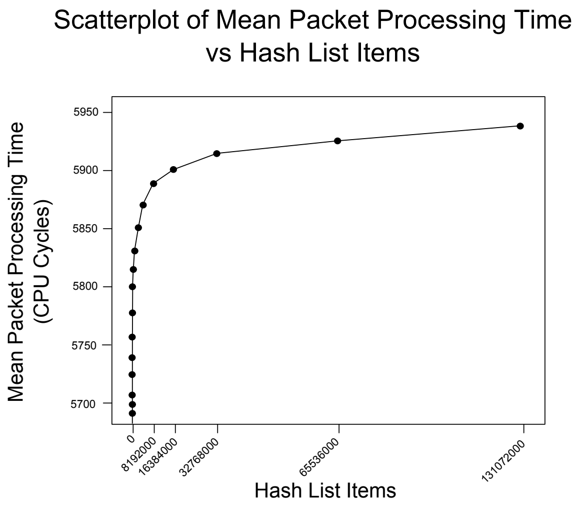

Figure 5. Mean Packet Processing Times versus 17 Different Hash List Sizes for Experiment 2.

To verify the logarithmic nature of the mean packet processing times as the hash list size is doubled, a separate plot is generated. Figure 6 plots the mean packet processing times against the natural log of the hash list sizes. The linearity of the plot asserts the logarithmic nature of doubling the hash list size.

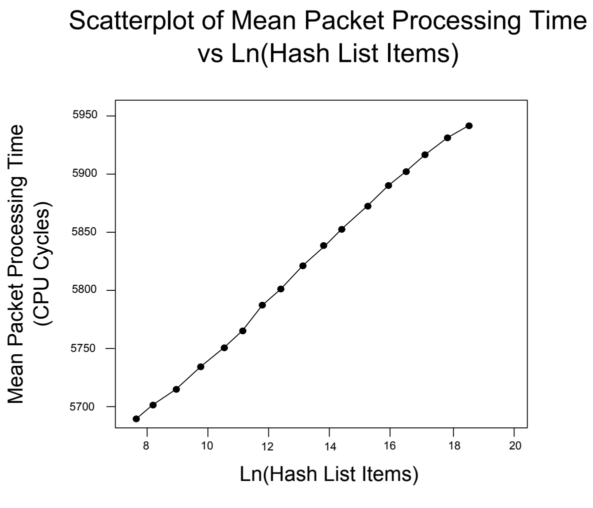

Figure 6. Mean Packet Processing Times versus Natural Log of 17 Different Hash List Sizes for Experiment 2.

Table 1 displays the mean packet processing times and the difference between them. For example, the hash list size of 4,000 takes an average of 13.84 more CPU cycles than the hash list size of 2,000. The range is from 6.82 - 23.55 CPU cycles and the overall average difference between the means is 15.93 CPU cycles. This results in the average addition of 15.93 CPU cycles to the overall packet processing time for each doubling of the hash list size which increases the packet processing time only slightly (an average of 0.27%).

Table 1. Difference Between Mean Packet Processing Times for 17 Different Hash List Sizes.

Hash List Items	Mean CPU Cycles	Difference Between Means
2,000	5683.87	-
4,000	5697.71	13.84
8,000	5707.06	9.35
16,000	5723.72	16.66
32,000	5739.69	15.97
64,000	5756.57	16.88
128,000	5780.12	23.55
256,000	5799.23	19.11
512,000	5814.21	14.98
1,024,000	5830.26	16.05
2,048,000	5848.29	18.03
4,096,000	5867.69	19.40
8,192,000	5886.57	18.88
16,384,000	5901.64	15.07
32,768,000	5918.90	17.26
65,536,000	5931.99	13.09
131,072,000	5938.81	6.82
	Average Difference Between Means	15.93

The four-byte sdbm hash contains eight hexadecimal values, e.g., 1F7B032A. Thus, there are a total of 4,294,967,296 (16^8) unique hashes for a four-byte hash. The maximum hash list size of 131,072,000 unique items for the TRAPP-2 system equates to 3.05% of the total number of hashes due to the 512 MB memory limit. If a system of 4,294,967,296 unique hashes is desired, then 16 GB of storage is required. The number 4,294,967,296 can be achieved by doubling the max list size of 131,072,000 approximately five more times. With a mean of approximately 16 additional CPU cycles per doubling of the hash list, a list of 4,294,967,296 unique hash items can be searched in additional 5 x 16 = 80 CPU cycles. This assumes the hash list is sorted to cater to the binary search algorithm. These results are encouraging for future research that will rely on larger hash list sizes.

5. Conclusions and Future Research

The first goal of this research is to determine the probability of packet intercept for a DNS packet of interest under various network utilizations. Experiment 1 reveals that the TRAPP-2 system captures, with 95% confidence, 91.89% of DNS packets of interest under a 93.7% network utilization (937 Mbps). The second goal of this research is to determine how increasing the hash list size affects the packet processing time. The original hash list size of 1,000 unique items is doubled 17 times to generate a hash list with 131,072,000 unique items. Experiment 2 reveals how each doubling of the hash list exposes the logarithmic nature of the packet processing time versus the number of hash list items. This is expected since the binary search algorithm is utilized. For this research, the binary search algorithm is chosen for simplicity. Implementing other data structures, such as a hash table, could result in faster hash lookups. Lastly, the mean packet processing time increases an average of 15.93 CPU cycles per doubling of the hash list size.

The first suggestion for future research is to expand the hardware capabilities of the Xilinx Virtex-5 ML510 FPGA board. The board contains an additional PowerPC processor and gigabit Ethernet controller that are not used in this research. Additional processing, functions, and algorithm work can be offloaded to the second processor. The second gigabit Ethernet controller can be used as a backup Ethernet controller. Future research can also focus on using a proven hashing algorithm such as SHA-1 or MD5 which use larger hash lengths. For this research, the sdbm hashing algorithm is selected because of its speed and simplicity. Shortfalls such as a minimal avalanche effect, potential collisions, and smaller hash length are overlooked. The algorithm processing can reside on a separate dedicated processor as mentioned above or on the FPGA. Thirdly, since a whitelist is used, the TRAPP-2 system is susceptible to a large number of false positives. Future research can look into limiting the number of false positives by sampling the DNS requests, coupling the TRAPP-2 with another security appliance, or inspecting the size and number of DNS requests per user. The last suggestion is to investigate how DNS Security Extensions (DNSSEC) and DNSCurve affect the detection of DNS tunneling.

References

[1] FOXNews, Report: Marine One Information Found on Computer in Iran (http://www.foxnews.com/politics/2009/03/01/reportmarine-information-iran/).

[2] Wired, Google Hack Attack Was Ultra Sophisticated - New Details Show (http://www.wired.com/threatlevel/2010/01/operation-aurora/).

[3] K. Schrader, An FPGA-Based System for Tracking Digital Information Transmitted Via Peer-to-Peer Protocols, Air Force Institute of Technology, March 2009.

[4] Xilinx, Virtex-5 Family Overview (http://www.xilinx.com/ support/documentation/data sheets/ds100.pdf).

[5] Oskar Pearson, DNS Tunnel - Through Bastion Hosts (http://archives.neohapsis.com/archives/bugtraq/1998_2/0079.html).

[6] Martin VanHorenbeeck, DNS Tunneling (http://www.daemon.be/maarten/dnstunnel.html).

[7] Kryo, Iodine by Kryo (http://code.kryo.se/iodine/).

[8] Dan Kaminsky, OzymanDNS 0.1 (http://www.doxpara.com/ozymandns_src_0.1.tgz).

[9] T.M Gil, NSTX (IP-over-DNS) HOWTO (http://thomer.com/howtos/nstx.html).

[10] N. Leidecker and A. Revelli, Introducing Heyoka: DNS Tunneling 2.0, (http://www.sourceconference.com/bos09pubs/Revelli-Leidecker_Heyoka.pdf).

[11] Ozan Yigit. sdbm - Substitute DBM or Berkeley ndbm for Every UN*X[1] Made Simple (http://cpansearch.perl.org/src/JESSE/perl-5.12.0-RC5/ext/SDBM_File/sdbm/README).

[12] Ozan Yigit, Hash Functions (http://www.cse.yorku.ca/∼oz/hash.html).

[13] Wireshark, Wireshark Network Protocol Analyzer (http://www.wireshark.org/).

[14] Snort, Snort FAQ (http://www.snort.org/snort/faq/).

[15] HiPPIE, About HiPPIE (http://hippie.oofle.com/about).

[16] D.A.L. Romana, S. Kubota, K. Sugitani, and Y. Musashi, DNS Based Spam Bots Detection in a University, *First International*

Conference on Intelligent Networks and Intelligent Systems, pp. 205-208, 2008.

[17] A. Karasaridis, K. Meier-Hellstern, and D. Hoeflin, Detection of DNS Anomalies using Flow Data Analysis, *Global Telecommunications Conference*, pp. 1-6, 2006.

[18] jhind, Welcome to the home of the meanypants projects (http://www.meanypants.com/meanypants/CatchingDNStunnelsWithAI-1.pdf?attredirects=0&d=1).

[19] Xilinx, Xilinx University Program Virtex-II Pro Development System (http://www.xilinx.com/products/devkits/XUPV2P.htm).

[20] Remote-Exploit, BackTrack (http://www.backtrack-linux.org/downloads/).

[21] Tcpreplay, Welcome to Tcpreplay (http://tcpreplay.synfin.net/).

[22] The Linux Foundation, pktgen (http://www.linuxfoundation.org/collaborate/workgroups/networking/pktgen).