An FPGA System for Detecting Malicious DNS Network Traffic

Brennon Thomas, Barry Mullins, Gilbert Peterson, Robert Mills

Chapter 15

AN FPGA SYSTEM FOR DETECTING MALICIOUS DNS NETWORK TRAFFIC

Brennon Thomas, Barry Mullins, Gilbert Peterson and Robert Mills

Abstract Billions of legitimate packets traverse computer networks every day. Unfortunately, malicious traffic also traverses these same networks. An example is traffic that abuses the Domain Name System (DNS) protocol to exfiltrate sensitive data, establish backdoor tunnels or control botnets. This paper describes the TRAPP-2 system, an extended version of the Tracking and Analysis for Peer-to-Peer (TRAPP) system, which detects BitTorrent and Voice over Internet Protocol (VoIP) traffic. TRAPP-2 is designed to detect a DNS packet, extract the packet payload, compare the data against a hash list and, if the packet is suspicious, log it for future analysis. Results show that the TRAPP-2 system captures 91.89% of DNS packets of interest under a 93.7% network load (937 Mbps). Also, as the hash list size is increased from 1,000 to 131,072,000 unique items, each doubling of the hash list size results in a mean increase of approximately 16 CPU cycles. These results demonstrate the ability of TRAPP-2 to detect traffic of interest under a saturated network load while maintaining large hash lists.

Keywords: Network forensics, DNS tunneling, detection system, FPGA

1. Introduction

Malicious network traffic continues to plague the Internet. Recent incidents include the blueprints for Marine One being leaked by a United States contractor via a file sharing program [3], and Chinese hackers pilfering intellectual property from Google and other United States companies [21].

As a result of these growing threats, the Tracking and Analysis for Peer-to-Peer (TRAPP) system [12] was developed to detect the use of the BitTorrent protocol and malicious content in the Session Initiation Protocol (SIP) used in VoIP telephony. The system resides on a Xil-

inx Virtex-II Pro FPGA board. It is limited by its 100 Mb Ethernet controller, small hash list size and inability to detect malicious DNS network traffic. However, TRAPP still captures packets of interest with a probability of intercept of at least 99% with a 95% confidence interval and 89.6 Mbps network utilization [12]. TRAPP is thus a viable network forensic tool that is worth expanding to incorporate a gigabit Ethernet controller, larger hash list sizes and malicious DNS detection.

This paper describes TRAPP-2, which extends TRAPP by incorporating a more powerful FPGA board and DNS protocol abuse detection. TRAPP-2 resides on a Xilinx Virtex-5 ML510 FPGA board with a faster processor and a gigabit Ethernet controller [17]. It captures 91.89% of DNS packets of interest under a 93.7% network load (937 Mbps). In addition, each doubling of the hast list size, from 1,000 to 131,072,000 unique items, results in a mean increase of approximately 16 CPU cycles.

2. Background and Related Work

This section discusses DNS tunneling, illicit traffic detection and the original TRAPP system, which sets the stage for the subsequent presentation of the TRAPP-2 system.

2.1 DNS 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 [9], which transfers non-DNS data in and out of a network via the DNS protocol. DNS tunneling is appealing because it is a covert channel and is operating system independent.

Figure 1 illustrates the concept of DNS tunneling. It involves the use of a hacker-controlled DNS server as an external trusted server to tunnel information out of a protected network using 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, the "infected" DNS traffic is allowed to pass. The data is 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. 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 five step process. 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

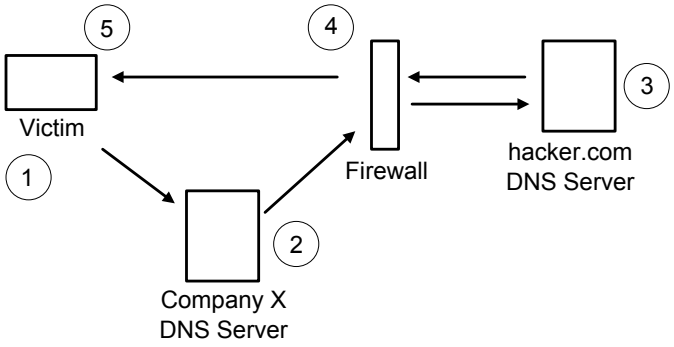

Figure 1. Establishing a DNS tunnel.

it requires Company X's DNS server to resolve the request. Company X's DNS server cannot resolve the request, so it forwards the request 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 as DNS is assumed to be trusted. The victim receives the DNS response to exfiltrate more data, connect to a botnet, etc.

Several DNS tunneling applications are available, including Iodine [8], OzymanDNS [6], NSTX (Nameserver Transfer) [4] and Heyoka [10]. We use Iodine to create DNS tunnels. Iodine offers benefits over other DNS tunnel implementations such as system portability, an MD5 challenge-response for login and the use of the NULL DNS record type to allow unencoded downstream data with up to 1 KB of compressed payload data [8].

2.2 Illicit Traffic Detection

Current methods for detecting illicit and malicious DNS traffic include signature-based software and statistical approaches. Software-based solutions include HiPPIE [2], Wireshark [16] and Snort [13]. All three solutions require more processing time because they operate at the application layer.

Malicious DNS traffic can also be detected using entropy-based systems. Romana, *et al.* [11] performed an entropy study of external DNS query traffic to a university's top domain server; peaks in entropy were assumed to be associated with spam botnet activity. The DNS Tunneling Attack Detector (TUNAD) [7] uses a similar technique to detect DNS packet size anomalies in real time. Finally, jhind [5] utilizes artificial neural networks to measure the entropy of previously captured `tcpdump` files. The disadvantage of entropy-based solutions is that they are generally unsuitable for real-time applications.

2.3 TRAPP System

The Tracking and Analysis for Peer-to-Peer (TRAPP) system [12] is an FPGA-based packet analyzer for detecting peer-to-peer protocols that transfer malicious content across a network. TRAPP was built specifically to detect BitTorrent and Session Initiation Protocol (SIP) packets. It was created as an alternative to current network traffic detection methods and is designed to operate at the gateway between the Internet and a local area network (LAN). It is placed on the switched port analyzer (SPAN) port of a switch, which means that a failure of TRAPP does not affect the network. This also makes TRAPP virtually undetectable to users.

TRAPP analyzes every packet flowing through the network switch in real time, looking for a BitTorrent or SIP signature. If a packet has a matching signature, TRAPP 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 BitTorrent file hashes or SIP URIs. If a match is found, the packet is logged; otherwise, TRAPP ignores the packet.

TRAPP suffers from several limitations. Its Xilinx Virtex-II Pro FPGA board incorporates a 100 Mb Ethernet controller and 300 MHz processor [18] that are suitable for smaller LANs; modern network bandwidth requirements necessitate the use of faster hardware. Also, TRAPP relies on 64 KB of memory to store the contraband hash list, which limits it to just 16,000 entries. Moreover, TRAPP is unable to detect illicit DNS traffic.

3. TRAPP-2 System

The TRAPP-2 system is developed and implemented on a Xilinx Virtex-5 FXT ML510 FPGA board. The FPGA implementation maximizes speed by allowing the software application to directly access the Ethernet controller buffer [12]. In addition, hardware and software modifications can be performed with minimal overhead. TRAPP-2 embodies some elements and functions from TRAPP. While both systems work in a similar manner, TRAPP-2 incorporates major hardware modifications to achieve the desired functionality.

The major hardware modification between TRAPP and TRAPP-2 is the Ethernet controller. TRAPP relies on the EthernetLite core peripheral, which has an upper limit of 100 Mbps. TRAPP-2 uses a Trimode Ethernet Media Access Controller that enables it to receive Ethernet frames at 1,000 Mbps. An accompanying 32,768-byte (maximum allowed) FIFO buffer stores Ethernet frames until they can be processed.

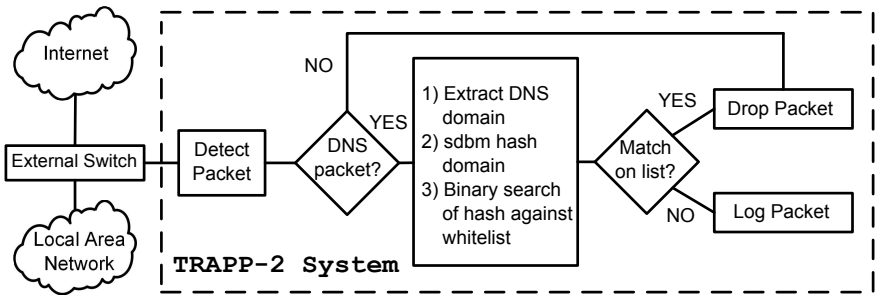

Figure 2. Packet data flow in the TRAPP-2 system.

The second hardware modification is the memory locations of the hash list and log file. The hash list contains a sorted list of hashes for determining if a DNS packet hash is of interest while the log file contains all the packets of interest that are detected by TRAPP-2. TRAPP relies on two sets of 64 KB block random access memory (BRAM) to separately store the hash list and log file. The maximum amount of BRAM available on TRAPP-2's FPGA is 128 KB per block. This limits the maximum hash list size, which is explored in Experiment 2 below. As a result, TRAPP-2 uses a 512 MB synchronous dynamic random access memory (SDRAM) scheme instead of the BRAM architecture to store the hash list and log file. Pilot tests reveal an average increase of 777 CPU cycles to detect and process a DNS packet using the SDRAM scheme. The 4,096-fold gain in physical memory address space at the cost of 777 CPU cycles is deemed to be acceptable. The memory configuration is also more realistic for future configurations that would require larger hash lists.

3.1 Algorithm

Figure 2 illustrates packet data flow in TRAPP-2. For DNS packets, TRAPP-2 detects a DNS request, extracts the entire domain, invokes `sdbm` (described below) to create a four-byte unique hash for the domain, compares the hash against a whitelist of approved domain hashes, and logs it if it is not in the DNS whitelist. A DNS request is defined as a UDP packet with a destination port of 53. Note that DNS zone transfers performed over TCP port 53 are not considered.

3.2 Hashing Function

The TRAPP-2 system implements the `sdbm` library hashing function [20]. The hashing function converts arbitrary-length strings (DNS do-

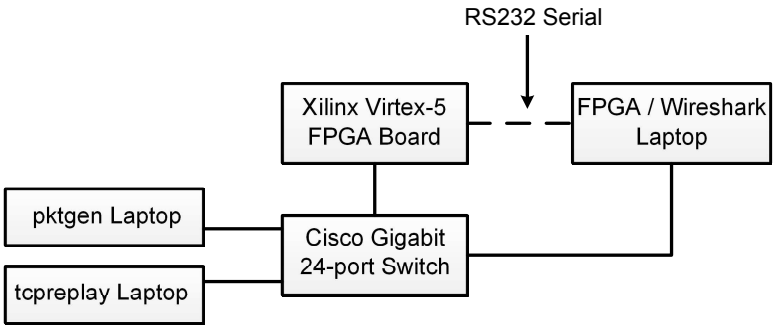

Figure 3. Experimental hardware configuration for the TRAPP-2 system.

mains) into four-byte uniform hashes to facilitate binary searches of the hash list. `sdbm` was selected over more proven hashing functions (e.g., SHA-1 and MD5) because it is quick and straightforward to implement [19]. One drawback with `sdbm` 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, which is not investigated in this paper. Pilot tests with `sdbm` reveal that an average of 86 CPU cycles are required to hash a six-character domain name and 1,195 CPU cycles are required for a 212-character domain name. This 86 to 1,195 CPU cycle increase in packet processing time is deemed to be acceptable.

4. Experimental Tests

Experiments were conducted to assess the performance of TRAPP-2. The hardware configuration used for the experiments is shown in Figure 3. It incorporates the following components:

- Cisco gigabit 24-port switch (model WS-C3560G-24PS-S) configured with 22 standard ports and two SPAN ports.
- Xilinx Virtex-5 FPGA board (model FXT ML510), which is connected to one of the SPAN ports on the Cisco switch.
- Dell Latitude D630 laptop loaded with the Windows XP Service Pack 3 Operating System. The laptop runs Wireshark 1.0.5 [16], which is connected to the other SPAN port on the Cisco switch; this acts as the control packet sniffer. The laptop is also used to program the FPGA via USB and to provide standard I/O for the FPGA board via a RS232 interface.

- Dell Latitude D630 laptop loaded with Backtrack 4 [1] and version 3.4.3 of `tcpreplay` [15] to inject packets into the network.
- Dell Latitude D630 loaded with the Ubuntu Desktop 9.10 Operating System and the Linux `pktgen` utility to create different network loads.

Two experiments were conducted. Experiment 1 was designed to determine the probability of packet intercept under various network loads. The probability of packet intercept was calculated by determining if a packet of interest was captured and successfully recorded in the log file. When measuring the probability of packet intercept, the network load of the system was also measured. The network load was equal to the total amount of traffic that entered TRAPP-2.

Experiment 2 was designed to determine how increasing the hash list size affects packet processing time. The packet processing time was measured as the number of CPU cycles required to process a packet. The PowerPC's System Timer timestamp function was used to tag when a packet arrived at the Ethernet controller and when the packet was completely processed.

The workload for TRAPP-2 consisted of a DNS packet and a network load. The malicious DNS packet was created using the DNS tunneling program Iodine prior to conducting the experiments. As mentioned above, the network load was generated using `pktgen`.

4.1 Experiment 1

Experiment 1 measured the probability of packet intercept of a DNS packet of interest while adding a non-DNS traffic load. 300 packets were sent at 200 ms intervals from the Backtrack laptop using `tcpreplay`. Injecting the packets at 200 ms intervals allowed for the result of each trial (captured or not captured) to be independent. Also, the sample size of 300 packets yields a good binomial distribution with small confidence intervals.

For each of the three replications, 300 packets were sent to TRAPP-2 and the number of packets captured were recorded. Before sending the 300 packets, five packets were sent to "warm up" the board by caching the data and instructions used by the processor.

Prior to injecting the packets, `pktgen` was activated to create a network load. `pktgen` permits the configuration of the packet size, number of packets and delay. The number of packets and packet size remained static at 6,000,000 packets and 1,500 bytes, respectively. The delay variable was modified to achieve the different network load percentages. A timestamp function within the BASH scripting language was used to

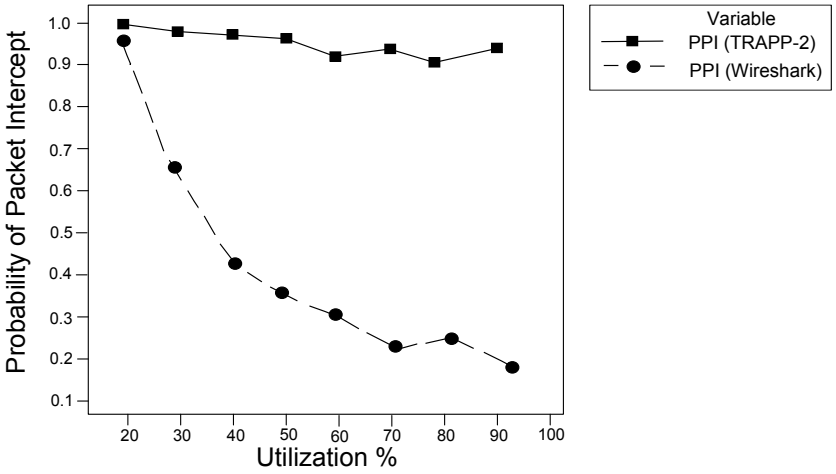

Figure 4. Packet intercept probability for DNS packets vs. network load.

record the number of nanoseconds since January 1, 1970. This timestamp function was taken immediately before `pktgen` was executed and immediately after completion. Since the total time required to send the 6,000,000 packets is known, and network load can be calculated. By adding the load, the resulting minimum network utilization was approximately 20% (204 Mbps) and was increased at 10% intervals up to the maximum achievable rate of 93.7% (equivalent to 937 Mbps).

Experiment 1 was performed under eight different non-DNS network loads. A total of 7,200 trials were involved: 1 packet type $\times$ 300 packets $\times$ 8 loads $\times$ 3 replications. A one-proportion confidence interval analysis was performed on the binomial variable to determine the probability of packet intercept and a 95% confidence interval for the proportion.

Figure 4 shows the probabilities of packet intercept for TRAPP-2 and Wireshark as the network load is increased. Confidence intervals were calculated, but they are too small and are virtually undetectable in the plot.

TRAPP-2 has a higher probability of packet intercept for every network utilization level. Figure 4 also reveals the approximate (and slight) linear decrease in the probability of packet intercept for TRAPP-2 as opposed to the exponential decrease for Wireshark as the network utilization increases. Moreover, TRAPP-2 manages to capture 91.89% of DNS packets at the maximum network utilization of 93.7%. In contrast, Wireshark only captures 18% of DNS packets at the maximum network utilization. The default buffer size of 1 MB used for Wireshark is significantly greater than the 32 KB FIFO buffer used in conjunction

with the FPGA's Ethernet controller. Increasing the buffer size in Wireshark could produce more favorable results, but the fact remains that TRAPP-2's buffer is smaller but still outperforms Wireshark.

4.2 Experiment 2

Experiment 2 examined how increasing the size of the hash list size affects the packet processing time. A series of 50 packets was sent from the Backtrack laptop using `tcpreplay`, which allows for sufficiently small confidence intervals to compare the results.

Three replications were performed. In each replication, 50 packets were sent and the number of CPU cycles required to process the packet was recorded. Prior to sending the 50 packets, five packets were sent to "warm up" the system. The network load in the experiment was limited to single packets injected into the system and was, thus, virtually zero.

To evaluate the effect of the hash list size on packet processing time, the hash list size was doubled from 2,000 up to 131,072,000 unique hash items, corresponding to seventeen different hash list sizes. The hash list was capped at 131,072,000 items because this uses 97.65% of the 500 MB of available memory.

Experiment 2 involved 2,550 trials: 17 list sizes $\times$ 1 packet type $\times$ 50 packets $\times$ 3 replications. A one variable t-test was performed to determine the mean packet processing time in CPU cycles, standard deviation, standard error of the mean, and 95% confidence interval for the mean.

Figure 5 shows a plot of the mean packet processing time as the hash list size increases. Once again, confidence intervals were calculated, but are not shown. The initial hash list size was 2,000 and the size was doubled to a maximum of 131,072,000. The doubling of the hash list size results in a logarithmic plot for the mean packet processing times. Note that the difference between the mean packet processing times for the minimum and maximum hash list sizes (2,000 and 131,072,000) is only about 255 CPU cycles.

Figure 6 shows a plot of the mean packet processing time against the natural logarithm of the hash list size. This verifies the logarithmic relationship of the mean packet processing time as the hash list size is doubled.

Table 1 presents the mean packet processing times for various hash list sizes and the differences between the mean values. Each doubling of the hash list size results in an average increase of 15.93 CPU cycles in the overall packet processing time.

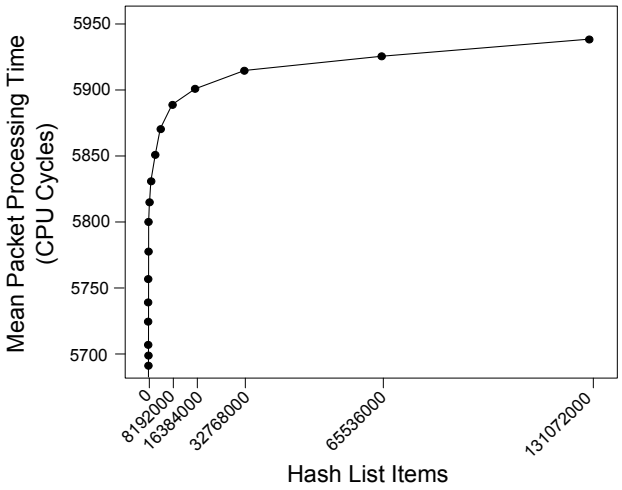

Figure 5. Mean packet processing times vs. hash list size.

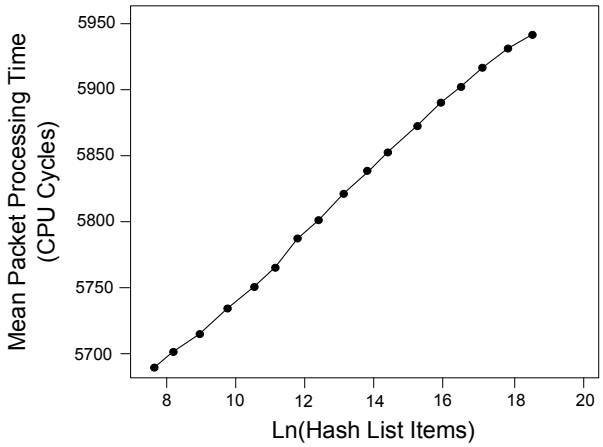

Figure 6. Mean packet processing time vs. natural log of hash list size.

The four-byte `sdbm` hash has eight hex digits (e.g., `1F7B032A`). Thus, there are a total of 4,294,967,296 (16^8) unique hash values for a four-byte hash. The maximum hash list size of 131,072,000 unique items for TRAPP-2 equates to 3.05% of the total number of hashes due to the 512 MB memory limit. With 16 GB of storage, the maximum size is 4,294,967,296 unique hashes. Since an average of 16 additional CPU cycles is required for each doubling of the hash list, a list of 4,294,967,296 unique hash items can be searched in an additional $5 \times 16 = 80$ CPU

Table 1. Mean packet processing times for various hash list sizes.

Unique Hash List Items	Mean CPU Cycles	Difference bet. 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		15.93

cycles. These results are encouraging for future implementations that could require larger hash list sizes.

5. Conclusions

TRAPP-2 extends the original TRAPP system by incorporating a more powerful FPGA board and DNS protocol abuse detection. Testing reveals that TRAPP-2 captures 91.89% of DNS packets of interest under 93.7% network utilization (937 Mbps) with 95% confidence. Also, the testing verifies the logarithmic relationship of the mean packet processing time as the hash list size is doubled. This is expected because a binary search algorithm is utilized to search the hash list. Implementing other data structures, such as a hash table, would result in faster hash lookups.

Future research involves using SHA-1 or MD5 hashes, which have longer hash values and fewer collisions than `sdbm`. TRAPP-2 is susceptible to high false positive errors because it employs a whitelist; future research will focus on limiting the number of false positives by sampling the DNS requests, coupling TRAPP-2 with another security appliance and inspecting the size and number of DNS requests per user. Also,

future research will investigate how DNS security extensions (DNSSEC) and DNSCurve would affect the detection of DNS tunneling.

References

[1] BackTrack Linux, BackTrack 4 (www.backtrack-linux.org).

[2] J. Ballard, HiPPIE (sourceforge.net/projects/hippie).

[3] FOX News Network, Report: Marine One information found on computer in Iran, New York (www.foxnews.com/politics/2009/03/01/reportmarine-information-iran/), March 1, 2009.

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

[5] jhind, Catching DNS tunnels with AI (www.meanypants.com/meanypants/CatchingDNStunnelsWithAI-1.pdf?attredirects=0&d=1).

[6] D. Kaminsky, OzymanDNS v. 0.1 (dankaminsky.com/2004/07/29/51), 2004.

[7] A. Karasaridis, K. Meier-Hellstern and D. Hoeflin, Detection of DNS anomalies using flow data analysis, *Proceedings of the IEEE Global Telecommunications Conference*, pp. 1–6, 2006.

[8] Kryo, Iodine (code.kryo.se/iodine).

[9] O. Pearson, DNS tunnel – Through bastion hosts (archives.neohapsis.com/archives/bugtraq/1998_2/0079.html), 1998.

[10] A. Revelli and N. Leidecker, Introducing Heyoka: DNS tunneling 2.0, presented at the *SOURCE Boston Conference* (www.sourceconference.com/bos09pubs/Revelli-Leidecker_Heyoka.pdf), 2009.

[11] D. Romana, S. Kubota, K. Sugitani and Y. Musashi, DNS based spam bots detection in a university, *Proceedings of the First International Conference on Intelligent Networks and Intelligent Systems*, pp. 205–208, 2008.

[12] K. Schrader, B. Mullins, G. Peterson and R. Mills, Tracking contraband files transmitted using BitTorrent, in *Advances in Digital Forensics V*, G. Peterson and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 159–174, 2009.

[13] Sourcefire, Snort, Columbia, Maryland (www.snort.org).

[14] The Linux Foundation, `pktgen`, San Francisco, California (www.linuxfoundation.org/collaborate/workgroups/networking/pktgen).

[15] A. Turner, `tcpreplay` (tcpreplay.synfin.net).

[16] Wireshark Foundation, Wireshark (www.wireshark.org).

[17] Xilinx, Virtex-5 Family Overview, San Jose, California (www.xilinx.com/support/documentation/data_sheets/ds100.pdf), 2009.

[18] Xilinx, Xilinx University Program Virtex-II Pro Development System, San Jose, California (www.xilinx.com/products/devkits/XUPV2P.htm).

[19] O. Yigit, Hash functions, Department of Computer Science and Engineering, York University, Toronto, Canada (www.cse.yorku.ca/∼oz/hash.html).

[20] O. Yigit, `sdbm` – Substitute DBM, The Guild of PD Software Toolmakers, Toronto, Canada (cpansearch.perl.org/src/JESSE/perl-5.12.0-RC5/ext/SDBM_File/sdbm/README).

[21] K. Zetter, Google hack attack was ultra sophisticated, new details show, Wired.com (www.wired.com/threatlevel/2010/01/operation-aurora), January 14, 2010.