Detection of DNS Anomalies using Flow Data Analysis

Anestis Karasaridis, Kathleen Meier-Hellstern, David Hoeflin

***Abstract*— The Domain Name System (DNS) is an essential network infrastructure component since it supports the operation of the Web, Email, Voice over IP (VoIP) and other business-critical applications running over the network. Events that compromise the security of DNS can have a significant impact on the Internet since they can affect its availability and its intended operation.**

This paper describes algorithms used to monitor and detect certain types of attacks to the DNS infrastructure using flow data. Our methodology is based on algorithms that do not rely on known signature attack vectors. The effectiveness of our solution is illustrated with real and simulated traffic examples. In one example, we were able to detect a tunneling attack well before the appearence of public reports of it.

***Index Terms*— Network Security, Anomaly detection, Relative Entropy, Denial of Service, DNS, Flow data.**

I. Introduction

THE Domain Name System (DNS) is an important network infrastructure service since it supports a number of critical applications such as the Web, FTP, Email and many other applications that require remote access to servers using the Internet Protocol (IP). The main function of DNS is to provide forward and reverse resolution of host/domain names to IP addresses. DNS is based on a distributed hierarchical protocol where groups of servers are responsible for certain domains and are dispersed around the world [1].

There are many known vulnerabilities in DNS [2]. Some are inherent to the protocol specification, while others stem from the protocol implementation or from server misconfiguration. For example, the protocol has only a rudimentary authentication mechanism that makes it an easy target of spoofing attacks, which can poison the cache and cause unauthorized redirection to illegitimate servers. Such events can cause significant problems such as denial of service, infections and "phishing". The most widely used software implementation of DNS is BIND, which is known to have many bugs that can be exploited in various ways to cause serious disruptions [3]. New vulnerabilities are often discovered in new versions of the software. Because of that, many organizations are open to attacks, if they do not continuously patch older versions or upgrade their software. Also, poorly planned deployment and configuration errors can open security holes, which if exploited, can cause havoc to critical services.

In this paper, we describe the algorithms used to detect DNS server cache poisoning and tunneling attacks over DNS. In Section II we give an overview of the DNS security monitoring prototype implementation, while Sections III and IV describe the cache poisoning and tunnel attack detector, respectively. We summarize our conclusions in V.

Anestis Karasaridis and David Hoeflin are with AT&T Labs. Kathleen Meier-Hellstern is with AT&T Government Solutions, Middletown, NJ 07748. This work was supported by AT&T Corp. Business Services, Government Solutions National Information Systems (NIS), Information Assurance (IA) under contract no. MDA904-03-C-1079

II. Overview

Our DNS security monitoring prototype software is divided into a number of modules each monitoring a different set of variables and employing a different set of algorithms to detect anomalies. Figure 1 provides an overview of the modules comprising the platform. The different modules process the data, calculate statistics, and employ algorithms to detect anomalies. If any of the modules detects an anomaly, it writes an alarm record to one of the Alert Reports. Other log files are used to provide more detailed information related to the alarms. The data flow and modules are replicated for each monitored link. The Alert Reports consolidate alarms from all monitored links.

The input data are unsampled flow records [4] collected at selected points in AT&T's network infrastructure. Each flow record contains the following information about the flow: Source IP address (*sip*), Destination IP address (*dip*), Souce Port (*sport*), Destination Port (*dport*), number of packets and bytes, start and end time of the flow, protocol, and the flags used (in the case where the protocol is TCP). Flow records summarize information about packets in one direction of a circuit (receive or transmit).

Below, we provide a detailed description of the function of two of the DNS monitoring modules, the cache poisoning detector and the tunneling attack detector.

III. Cache Poisoning Detector

Cache poisoning refers to the act of changing the contents of a server's cache to alter the correct mapping between host names and IP addresses. The objective of such an attack is to redirect end-user requests to illegitimate servers controlled by the attackers. This problem exists in DNS because the protocol allows caching without applying a strong authentication process to verify the authenticity of the source of a response.

As described in detail in [5], a particular type of cache poisoning attack, called 'birthday attack', can be launched against a DNS server by simply sending one or more requests for a domain such as `http://www.att.com` followed shortly by maliciously crafted responses that contain as an answer the IP address that the attacker wants to poison the cache with. This IP address would not be the true address of the hostname/domain `www.att.com` but one chosen by the attacker to redirect legitimate traffic to. The responses usually

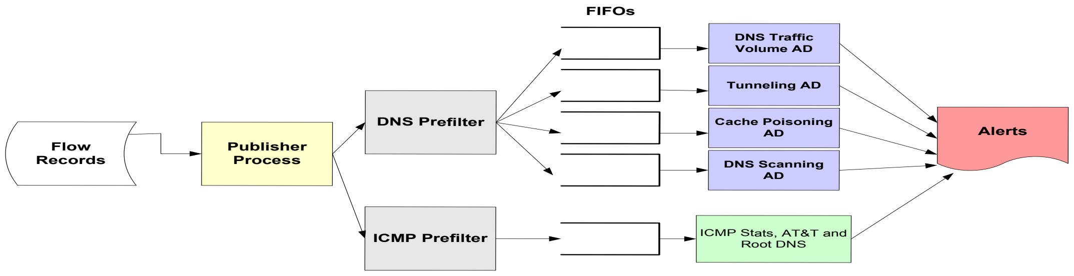

Fig. 1. Overview diagram of the DNS security monitoring system

have a spoofed source IP address and a random transaction ID that the attackers try to match by brute force with the request's transaction ID. DNS uses the source IP address and the transaction ID as a way to authenticate the originator of the responses and to match the response with a request. Therefore, a DNS server can be easily fooled by using a spoofed source IP address and a transaction ID in the response that matches that of the request. There are various techniques and free software available to launch a cache poisoning attack.

Since the transaction ID of requests or responses is not available in the metadata records, we use other information to detect an attack such as the IP addresses of sources and destinations, the port numbers, the interarrival times of packets and the sequence of events that take place. At a high level, the algorithm works as follows:

- For each DNS flow, identify if the packets in the flow are requests or responses (based on port numbers).
- If the packets in the flow record are requests, combine the flow with a thread[1] of request flows that have the same destination IP (*dip*) and the same bytes-per-packet ratio (*bpr*), if the flow arrives within a small time interval (e.g., 1sec) after the last flow in the thread. If the thread exceeds a certain number of flows (e.g., 50), summarize the information in the flow records and provide an alert that repeated requests took place.
- If the flow record contains response packets, combine the incoming flow with similar flows by *dip*, *bpr*, destination port (*dport*) and source IP (*sip*) in the event that the flow arrives within a small time interval (e.g., 1 sec) since the arrival of the last flow in the same thread. Examine if the number of such flows exceeds a threshold (e.g., 50). If the threshold is exceeded, examine all the request threads for the same *dip* and see if they have a *bpr* such that
$$bpr_{response} > bpr_{request} + bpr_{THR}, \qquad (1)$$
where bpr_{THR} is the minimum number of additional bytes (set at 16) required to construct a response for a given request[2]. Also, examine if the response thread starts within the time limits of the request thread. If all the above conditions are true, summarize the flows and generate an alarm for cache-poisoning attack. If the number of responses in the thread is larger than a threshold, but it is not within the time frame of a request thread or it cannot be matched to a request thread because the *bpr* condition (1) is not met, then the algorithm produces an alarm for repeated responses.
- Empty and restart request and response threads when flows come much later (e.g., more than 1 sec later) or earlier than the last flow in the thread.
- Periodically (e.g., every 500,000 input flows) clean up threads to free up memory

We tested the accuracy and effectiveness of the algorithm using synthetically generated data that simulate a cache poisoning attack. A script was written to generate the attack flows (metadata flows with the same format and fields as the original metadata). Then, the attack flows were merged into field data keeping the original time sequence. A sample of the flows that were used to simulate a cache poisoning attack is shown in Table I. The detection module accurately captured the attack. Figure 2 shows the output of the cache poisoning detection algorithm, containing the alarm messages.

The alarm messages indicate three levels of severity (number 1 being the most severe):

1) Severity 1: "Suspicious Responses": In this case, the repeated responses have been matched to the repeated

[1]Thread here denotes a group of objects such as flows with similar characteristics.

[2]In DNS the response contains the original request, therefore the response packet is larger than the original request packet by a specific number of bytes.

TABLE I

SAMPLE FLOW RECORDS THAT ARE USED TO SIMULATE A CACHE POISONING ATTACK. PACKET TYPE 1 IS DNS REQUEST WHILE TYPE 2 IS A DNS RESPONSE. IP ADDRESSES HAVE BEEN MASKED FOR PRIVACY PROTECTION.

Packet Type	Source IP	Destination IP	Packets	Bytes	Start Time	End Time	Source Port	Destination Port	Protocol
2	20.20.20.30	5.10.15.20	1	90	1066831077	1066831077	53	51453	17
1	20.30.10.5	5.10.15.20	1	70	1066831078	1066831078	14414	53	17
2	20.20.20.30	5.10.15.20	1	90	1066831078	1066831078	53	51453	17
1	100.50.25.25	5.10.15.20	1	70	1066831079	1066831079	50235	53	17
1	200.100.50.25	5.10.15.20	1	70	1066831079	1066831079	65164	53	17
1	50.100.150.200	5.10.15.20	1	70	1066831079	1066831079	61480	53	17
1	10.20.30.40	5.10.15.20	1	70	1066831079	1066831079	10105	53	17

```
Repeated Requests:dip=5.10.15.20,
number=307,bpr=70,link=SiteA,
sip=var,sport=var,
start_time=Wed Oct 22 13:57:58 2003,
end_time=Wed Oct 22 14:00:38 2003

Suspicious Responses:dip=5.10.15.20,
number=811,bpr=90,link=SiteA,
sip=20.20.20.30,dport:51453,
start_time=Wed Oct 22 13:57:57 2003,
end_time=Wed Oct 22 14:05:00 2003

Repeated Responses:dip=5.10.15.20,
number=811,bpr=90,link=SiteA,
sip=20.20.20.30,dport:51453,
start_time=Wed Oct 22 13:57:57 2003,
end_time=Wed Oct 22 14:05:00 2003
```

Fig. 2. Sample output of the cache poisoning detection algorithm.

requests and the confidence that an attack is currently taking place is very high.

2) Severity 2: "Repeated Responses": The algorithm has identified repeated responses to the same target and destination port but it was not able to match them to requests, possibly because the requests came through a different link.
3) Severity 3: "Repeated Requests": Repeated requests have been detected but they were not matched to repeated responses. This is a more common event in the Internet today and can be the indication of other attacks such as Denial of Service (DoS) attacks. It can also be the result of misconfigured or misbehaving DNS clients or servers that are used for other illegitimate/illegal purposes such as sending spam or "phishing" email.

For each type of alarm, the software generates a summary line which gives the target IP, the number of repeated messages in the thread, the common *bpr* in these messages, the link that the pattern was detected on, the source IP (or "var" if it varies), and the start and end times of the activity.

IV. TUNNELING ATTACK DETECTOR

The purpose of the Tunneling Attack Detector (TUNAD) is to detect in near-real-time anomalies in the packet size statistics of traffic over the typical DNS port. Such changes are indicative of wide scale suspicious tunneling activity over DNS.

```
#Stats FROM: Wed Dec 31 23:59:33 2003
#      TO: Thu Jan  1 00:59:43 2004
#Total DNS records: 433024
#Request size, Request size frequency(%)
32 0.0149
33 0.0004
35 0.2451
36 0.1397
39 0.0051
...
```

Fig. 3. A sample of an hourly packet size statistics file. Lines that start with # are comments that provide relevant information about the statistics.

The first step taken by TUNAD is to separate DNS packets to requests (when the source port $sport > 1023$ and the destination port $dport = 53$), responses ($sport = 53$ and $dport > 1023$) and packets whose query type is initially unknown (*uqr*), i.e., messages for which we do not know with certainty if they are requests or responses ($sport = 53$ and $dport = 53$). In order to measure the exact packet size from flow data, we use single packet flow records (flow records that contain information about a single packet). After separating the traffic, TUNAD calculates hourly packet size histograms for each monitored circuit and each packet type (requests, responses and *uqr* packets). Figure 3 shows a small part of an hourly statistics file.

In addition, for each of the packet types, we calculate the frequency of the packets that have byte sizes not conforming to the normal UDP/DNS packet size requirement (we call these non-conforming packets). For responses and *uqr* packets, we calculate the percentage of packets that exceed 512 bytes in size (this size includes the Layer 3 header but no headers for any layers below this). Recent extensions of the DNS protocol allow queriers to specify to responders that they can process DNS UDP packets larger than 512 bytes using the OPT Resource Record [6]. This means that we would expect normally some DNS UDP packets to exceed the 512 bytes limit. Our purpose is to detect changes in the frequency of large packets. For requests we calculate the frequency of packets with sizes larger that 300 bytes (largest request for a single host query is not expected to exceed this size [7]). A sample output of the frequencies of non-conforming packet sizes is shown in Figure 4.

Once, the statistics for the last processed hour are computed for each packet type, the packet size anomaly detectors are launched. We use a Cross-Entropy based anomaly detector on

```
#Date:Hour  Req_ratio Rsp_ratio
20031223:18 0.71663 0.30041
20031223:19 0.80219 0.30520
20031223:20 0.78419 0.30866
20031223:21 0.79153 0.32586
20031223:22 0.87635 10.41829
20031223:23 1.02574 0.29167
20031224:00 1.00535 2.35654
...
```

Fig. 4. Sample contents of a file that contains frequencies of non-conforming packets.

the packet size histograms and a Cusum based detector on the frequencies of non-conforming packets.

In the following section, we give a description of the Cross-Entropy based algorithm to detect anomalies in DNS packet sizes.

A. Cross-Entropy Anomaly Detection

We use Cross-Entropy [8] to detect significant changes in the distribution of conforming and non-conforming packet sizes. Given a baseline distribution defined by the probabilities q_i, and an observed distribution defined by the probabilities p_i, Cross-Entropy E_c is defined as

$$E_c = -\sum_i p_i \log(q_i). \tag{2}$$

In the current application, p_i is the probability of seeing a given packet size in bin i and q_i is the probability of seeing the packet size in the same bin in the baseline data. Each bin is assigned to an individual packet size except for packet sizes larger than 512 and smaller than 28 bytes (usually non-conforming packet sizes), which are assigned to a single bin. E_c is a measure of how much the distribution has changed: If for example, we currently see packets sizes in bin i that we have rarely seen in the past, p_i would be relatively large, and q_i would be close to zero, making the product $-p_i \log(q_i)$ very large and therefore increasing significantly the value of E_c in (2). If, on the other hand, the distribution has not changed, then Cross-Entropy becomes equal to Self-Entropy, E_s which is defined as in (2), where q_i is replaced by p_i. In theory, Cross-Entropy is always greater than or equal to Self-Entropy, and we use their difference

$$E_r = E_c - E_s = \sum_i p_i \log(\frac{p_i}{q_i}) \tag{3}$$

as a measure of change in the distribution. This distance function is also referred as *Kullback-Leibler distance* or *Relative Entropy*. The algorithm generates alarms when the relative entropy E_r exceeds a certain threshold THR_{Alert}. This threshold is tunable and can be set at a level that indicates a significant change in the behavior of the traffic, such as the one seen during the spread of the Sinit virus [9]. Figure 5 shows the Cross, Self and Relative Entropy calculation for one of the monitored links around the onset of the Sinit virus. The algorithm indicated a significant change in the Relative Entropy of the packet sizes on Sept. 30, 2003, before the first external reports of suspicious activity surfaced on Oct. 2, 2003.

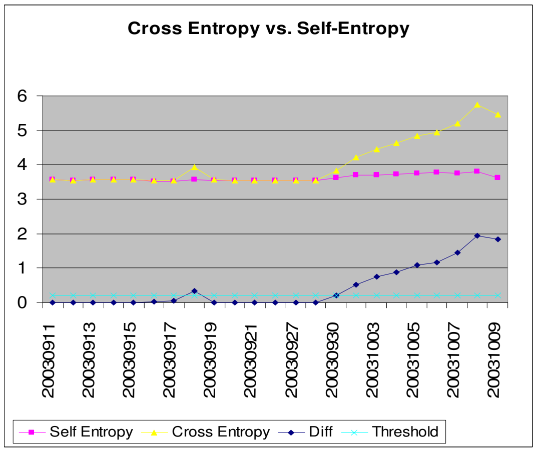

Fig. 5. Cross, Self and Relative Entropy are plotted for data collected on the `SiteA` circuit around the onset of the Sinit virus. The calculation of the Entropy measures was based on daily packet-size statistics.

These changes were attributed to the fact that port 53 was not being used exclusively to carry DNS traffic.

The baseline probabilities are calculated for each hour of the day to take into account time-of-day effects, while the observed probabilities are compared to the ones in the same hour of the baseline. In order to calculate the initial baseline, a set of baseline days can be initially specified in a file. The probabilities of each packet-size bin in each hour are calculated as the average of the corresponding probabilities in the set of baseline days. The baseline probabilities are updated adaptively as new data are processed. Our objective is to have the algorithm adapt to very small changes in the Relative Entropy E_r, which are expected under normal conditions as new hosts, applications, operating systems and DNS servers start generating traffic that changes the overall statistics. The algorithm tracks the number of hours in a day that the distance E_r is relatively small (threshold can be set for example as one half of the alarm threshold level), and if this number reaches 24, then it adds the current day into the set of days used for the baseline. If the number of days used in the baseline is already at the configured maximum (e.g., seven), then the earliest day is dropped and the new baseline is recalculated. If a single hour in the day does not satisfy the above criterion, then the day is not considered as part of the baseline.

The algorithm is also designed to keep track of changes in the overall level of the Relative Entropy that stabilize over time and to give feedback to the analyst that a new level has been established. This capability was incorporated because certain events, such as routing changes or power outages, can cause a significant (above the alarm threshold) rise in the distance from the baseline, that can remain high for a long period of time. In order to implement this capability, the algorithm calculates the average slope (first derivative) of E_r within a configurable window of time W. If the average slope within W is close to zero (or some very small positive or negative number)

$$\left|\overline{\frac{dE_r}{dt}}\right| < \epsilon, \tag{4}$$

where ϵ is a small positive number, and there is no significant disturbance in the distance,

$$\left|\frac{dE_r}{dt}\right| < d_{THR}, \forall t \in W, \tag{5}$$

where d_{THR} is configurable, then the distance is considered stable, and the analyst is given the option to apply a manual override to update the baseline. This also allows time for the analyst to drill down into the data pointed by the alarms. A manual overide of the baseline will reduce alarms that do not provide additional value. The algorithm ensures that there is complete data available for each hour for at least a certain percentage (currently set at 60%) of the days included in the time window W. The derivative values are calculated by taking the one-step differences of the individual E_r values. Parameters ϵ, d_{THR} and W can for example be set at $5e-4$, $Alert_{THR}/4$ and 48 hours, respectively.

Validating the accuracy of the algorithm in detecting security compromises is challenging since there is no availability of data with known attacks. Instead, we monitor for external reports in the Internet, which refer to events that have common characteristics with the ones detected by our algorithm. One such central repository of reports is the Internet Storm Center (ISC) [10]. ISC aggregates alarms from many Intrusion Detection Systems (IDS) around the world to one location and provides summary data, such as the numbers of suspicious sources and targets that appear in the alarms. Changes in the counts of such sources and targets are indicative of unusual activities. The activities are broken up by port, so normal or tunneling DNS activities can be monitored by looking at port 53. We also look for publicly available DNS packet content data that is the cause of IDS alarms. These external reports are evaluated along with packet content data that are collected internally.

Figure 6 illustrates the ISC data during the period between November 13, 2003 and January 24, 2004. The data indicate significant activity during the periods of November 17-27, 2003 and from December 9, 2003 to January 24, 2004. For a comparison, the average number of suspicious sources and targets during the quiet period between 1/1/2003 and 9/29/2003 were approximately 3300 and 2000 per day, respectively. Therefore, the levels of suspicious sources and targets in the period in question were about 30 to 40 times the average levels during the quiet period. We used the period between 11/13-15 to calculate the initial baseline and then calculated E_r from 11/16 to 1/24, as illustrated in Figure 6. Results are presented from four monitored circuits. One of the circuits was upgraded at the end of December and split into 4 circuits. Since the split we monitor the 'receive' channel of that circuit.

Figure 7 illustrates that our algorithm detected a strong signal between 11/16 and 11/20 on the `SiteC` circuit and from 12/02 to 1/19/04 on the `SiteC` and `SiteA` circuits. These periods are around the times that the ISC reports show

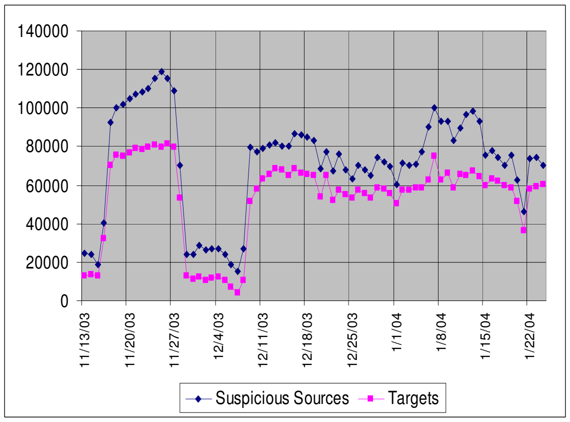

Fig. 6. Internet Storm Center (ISC) data for port 53 during the period between 11/13/2003 and 1/24/2004. The data illustrate the number of suspicious sources and targets that appear in Intrusion Detection Systems (IDS) alarms from around the world. A high number is indicative of increased suspicious activity over the given port.

increased activity. The algorithm also picks strong anomalies on the `SiteD` and `SiteB` circuits.

Our analysts found that there is a link between the suspicious DNS activity and the propagation of the Sinit trojan that uses port 53. According to the analysis of the trojan communication protocol [9], Sinit uses a new paradigm to spread and update its code, which has similarities to a Peer-to-Peer network. The spread of the Trojan gained more publicity by a New York Times article on December 8th, 2003 [11].

1) Selection of Alert Thresholds: In order to select the alert threshold THR_{Alert}, one could take a statistical approach using the fact that the KL distance as a difference of log likelihoods converges to a chi-square distribution under the null-hypothesis. Thus a threshold can be set at $\chi^2_{1-\alpha,c}/(2N)$, where α is the false positive rate, c is the number of classes (or bins) and N is the number of samples. For example for $c = 500$, $\alpha = 10^{-3}$, then $\chi^2_{1-\alpha,c} \approx 603$ (if $\alpha = 10^{-4}$, then $\chi^2_{1-\alpha,c} \approx 626$). Hence for large N in the order of 10^4 the threshold is about 0.03. Large sample sizes allow very sensitive tests, i.e., we can detect very subtle changes in the distribution. In this application however, it is clear that subtle changes are not necessarily the ones we are most anxious to detect. Setting the threshold to such a low value could overwhelm the investigation team with too many alarms when the distributions are statistically different but not of interest.

Alternatively, intuitively one can think of two distributions f and g as being either close if the number of observations that must be taken to distinguish between the two is large, or far apart if the number required is small. Formally, we have for given type I (false positives) and type II (false negatives) errors α and β respectively, via Kullback's equations [12, pp.74], that

$$E(n|f) \int f \ln(f/g) \geq \beta \ln(\frac{\beta}{1-\alpha}) + (1-\beta) \ln(\frac{1-\beta}{\alpha}) \tag{6}$$

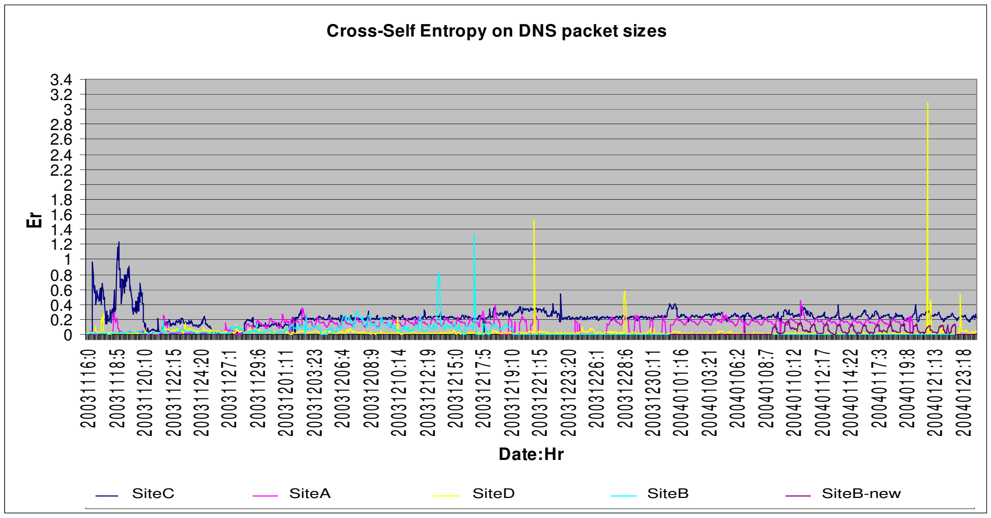

Fig. 7. Calculation of the Relative Entropy E_r with adaptive baseline updates during the period between 11/16/2003 and 1/24/2004. The days 11/13-15 were used to calculate the initial baseline for all circuits.

and

$$E(n|g)\int g\ln(g/f)\geq\alpha\ln(\frac{\alpha}{1-\beta})+(1-\alpha)\ln(\frac{1-\alpha}{\beta}),\quad(7)$$

where $E(n|d)$ is the expected sample size given the distribution d is correct.

From an application point of view, we are interested only in those cases where f and g differ greatly, that is, where the expected number of observations required to distinguish them is small, e.g., to a rough order of magnitude 10 rather than 100 or 1000. For $\alpha = 10^{-2}$, $\beta = 10^{-3}$ and $E(n|d) = 10$, then

$$\int f\ln(f/g)\geq 0.46\quad(8)$$

and

$$\int g\ln(g/f)\geq 0.68\quad(9)$$

Hence we recommend using threshold values in the range of .5 to .7 as the threshold for the KL distance when deciding a distribution is different enough from the baseline to warrant closer investigation.

V. Summary

We described algorithms and a prototype system to detect attacks over the Domain Name System (DNS) using flow-level data captured in a large Tier-1 ISP network. Our prototype system monitors and detects large security events such as scanning and DoS attacks, as well as under-the-radar types of attacks such as cache poisoning and tunneling attacks using DNS ports. In this paper, we presented our approach in detecting the latter types of attacks, which are generally more difficult to track. We illustrated the effectiveness of our approach through simulations and real network events.

References

[1] P. Mockapetris, "Domain names - concepts and facilities," IETF, Request for Comments (RFC) 1034, 1987.

[2] R. Chadramouli and S. Rose, "Challenges in securing the domain name system," *IEEE Privacy and Security Magazine*, January/February 2006.

[3] Internet Systems Consortium, "Bind vulnerabilities," http://www.isc.org/index.pl?/sw/bind/bind-security.php, 2005.

[4] Cisco Systems, "Cisco IOS Netflow," http://www.cisco.com.

[5] J. Steward, "DNS Poisoning - The Next Generation," LURHQ Inc., Tech. Rep., 2002.

[6] P. Vixie, "Extension Mechanisms for DNS," IETF, Request for Comments (RFC) 2671, 1999.

[7] P. Mockapetris, "Domain names - implementation and specification," IETF, Request for Comments (RFC) 1035, 1987.

[8] P-T. De Boer, D.P. Kroese, S. Mannor, R.Y. Rubinstein, "A tutorial on the cross entropy method," *Annals of Operations Research*, 2005.

[9] Threat Intelligence Group, "Sinit p2p trojan analysis," http://www.lurhq.com/sinit.html, December 2003.

[10] Internet Storm Center, "Trends," http://isc.sans.org, 2004.

[11] John Schwartz, "Hackers steal from pirates, to no good end," New York Times, December 2003.

[12] S. Kullback, *Information Theory and Statistics*. Dover Publications, 1968.