

Malware Beaconing Detection by Mining Large-scale DNS Logs for Targeted Attack Identification

Andrii Shalaginov, Katrin Franke, Xiongwei Huang

Abstract—One of the leading problems in Cyber Security today is the emergence of targeted attacks conducted by adversaries with access to sophisticated tools. These attacks usually steal senior level employee system privileges, in order to gain unauthorized access to confidential knowledge and valuable intellectual property. Malware used for initial compromise of the systems are sophisticated and may target zero-day vulnerabilities. In this work we utilize common behaviour of malware called "beacon", which implies that infected hosts communicate to Command and Control servers at regular intervals that have relatively small time variations. By analysing such beacon activity through passive network monitoring, it is possible to detect potential malware infections. So, we focus on time gaps as indicators of possible C2 activity in targeted enterprise networks. We represent DNS log files as a graph, whose vertices are destination domains and edges are timestamps. Then by using four periodicity detection algorithms for each pair of internal-external communications, we check timestamp sequences to identify the beacon activities. Finally, based on the graph structure, we infer the existence of other infected hosts and malicious domains enrolled in the attack activities

Keywords—Malware detection, network security, targeted attack.

I. INTRODUCTION

TARGETED Attacks are a type of threat in which actors actively pursue and compromise a target entity's infrastructure, while preserving anonymity [1]. In contrast to Advanced Persistent Threat (APT) attacks, targeted attacks are not carried by states; they have a rather narrow scope and are performed by a community of attackers. These attackers have a certain level of expertise and have sufficient resources to conduct their activities over a long-term period. They can adjust their attacks to counter the victim's defence. APT and Targeted Attacks have clearly proven themselves capable of penetrating common security solutions like anti-virus, intrusion detection systems and endpoint protection software [2]. Moreover, they can remain undetected for months at a time, all while siphoning off valuable data or carrying out destructive actions. Using the Mandiant Compromise assessment schema [3], attacks on networks often consist of several main stages: *Initial hosts compromise, Downloading additional payload, Command and Control (C2) communication, Steal target data*. So, installation of a conventional commercial anti-virus solution on a system is

not a sufficient safeguard against attack; it is also necessary to incorporate network-related indicators of malware infections. Current signature-based security solutions are only capable of handling known threats that are based on the artefacts found during malware sample analysis. Knowing this to be the case, attackers are resorting to ever more sophisticated techniques to evade detection and maintain persistence. Behaviour based detection on network events might be able to unveil some malicious activities unknown to traditional static analysis methods, but it is still susceptible to evasive measures. In this work we consider *C2 communication* as a key indicator of compromised systems.

The first objective of malware execution used in targeted attacks is to exploit a specific vulnerability and establish a privileged process in the target system. This process downloads an additional malicious payload, which will then attempt to contact the malicious C2 [4]. Additionally, propagation of the access compromise beyond the target system requires opening an external communications channel to the C2 server. This behaviour will leave a record of itself in network flow and DNS logs, which provides us with a chance to identify the infected internal hosts and external malicious domain names. Analysis of malware that were previously identified in targeted attacks has uncovered common behavioural characteristics exhibited by the infected hosts' "beacons". According to Trend Micro research in detection of APT [5] the *beacon* in network traffic is (are) communication packet(s) sent inside of the network at regular intervals, which may be found using DNS requests or URLs. *Beacons* can be used for a variety of purposes such as obtaining new tasks from a C2 server, downloading updates, etc. To evade detection, almost all malware will take some measures to hide their footprint. In the compromised networks, nearly all APT malware uses some form of obfuscation on the outbound callback [5]. Additionally, they will not use common blacklisted domains like ".cn" or ".ru". Furthermore, they do not fast-flux through the IP addresses. In many cases, attackers use rented servers in legitimate data centers. They can also use exploits deployed on whitelisted websites for the first hop in C2. Such communication obfuscation is often not a standard algorithm that can be decoded, but rather a proprietary or an embedded steganographically into benign objects. One example is the WEBC2 [6] backdoor family used in an APT attack, which is capable of downloading and executing a payload. Then, it attempts to communicate with its C2 once a week (for example, Thursday at 10:00

A. Shalaginov, K. Franke, and X. Huang are with the Testimon Research Group, Norwegian Information Security Laboratory, Faculty of Computer Science and Media Technology, Gjøvik University College, Teknologivegen 22, P.O.Box 191, N2802 Gjøvik, Norway (phone: +47 61 13 53 75 ; e-mail: andrii.shalaginov@hig.no).

AM). Command Five Pty Ltd [7] presented a report on an investigation into several Advanced Thread Attacks in 2012, where they identified some malware that attempt to communicate with their C2 infrastructure at frequent intervals. This implies that beacon interval can reveal such regularities.

This paper is aimed at periodicity detection in timestamps from large-scale enterprise DNS log data for malicious communication detection. We pursue several goals: (i) Timestamps analysis in DNS log files to identify malicious beacons, (ii) Revealing potentially infected individual hosts and the full scope of attack events based on the detected beacons, (iii) Feasibility study of application of periodicity algorithms for beacon detection with respect to speed and obfuscation resilience. We also believe that higher frequency beacon detection is enabled by their consistent in time series. However, there are multiple challenges that are described below. In addition, we implemented the experiments based on Graphlab Create [8], which achieved near real time processing and detection ability [9]. In contrast to Opreal et al. [10], we present to apply periodicity detection, rather than a belief propagation framework from graph theory. Using such an approach, we are able to achieve high accuracy and computational speed.

The paper is organised as follows. Section II presents the state of art in malware beaconing. Section III explains our targeted attack detection methodology, including data pre-processing, the beacon detection module and the inference module. Afterwards, the experiments are introduced in Section IV which is followed by an analysis of the results in Section V. Conclusions and discussions are given in Section VI.

II. CHALLENGES WITH ZERO-DAY MALWARES AND TARGETED ATTACKS DETECTION

Network beaconing activity is prevalent in many applications and protocols. For example, Network Time Protocol (NTP) [11], Rich Site Summary (RSS) feeds [12], automated software patching or updating that keep alive traffic in long lived sessions may also appear as beacons. Therefore, most of beacons are not malicious, while malicious beacons are sourced from infected hosts where the malware repeatedly attempts to establish remote connectivity with a malicious C2 server. Since malicious beacons only account a tiny part of all beacons, this leads to a high false positive rate for malware beacon detection methods. Additionally, not all malware exhibits beacon behaviour during the infection process. Some malware agents will randomise their communication intervals, emulating legitimate communications behaviour. Others may take advantage of multiple channels to hide by using a single unique channel for a short period of time as described by van Duijn [13], where PCAP files were used to detect beacons. We do not consider malware with cloaking since this is out of the scope of the paper. We believe that periodicity detection, even obfuscated, may be a strong indicator of malware.

A. C2 in Malware Control

First, we need to understand how the beacons are created and under which circumstances. Gu et al. [14]

studied two common methods of C2 communications: *Push* and *Pull*. *Push* describes an active communication model in which an attacker controls compromised hosts directly through C2 server, while *Pull* refers to a passive mode, in which the server periodically sends jobs to an attacker and later retrieves the results. Furthermore, OpenDNS presented following topologies in early 2000, commonly found in C2 architectures [15]: *Centralized*, before 2003, includes a single C2 server controlled by an attacker based on IRC or HTTP/HTTPS protocols, *Distributed*, before 2006, binds P2P peers that are able to communicated between each other and C2 server, *hybrid* that are currently in use (like Zeus), combines functionality of two previous topologies to create more resilient and advanced model. According to the white paper, DNS tunnelling has recently become popular in *hybrid* architectures and it is considered to be relatively unexplored area.

B. Case Study of Malware Beaconing

To understand the work mechanism of a beacon, we refer to the user guide of beacon payload examples by Cobalt Strike [16], which is a commercial product for modelling APT and targeted attacks. For example, it can use the SMB protocol to create a beacon that can also use protocols like HTTP or DNS. Even through the concrete techniques used in target attacks varies; the behaviour indicates similar properties, which could let us understand beacons from a broader point of view. The initial task is to start a beacon listener to use malicious payload and specify the port number which used to transmit traffic, the payload provides two communication channels. After a list of domains are provided as described in Fig. 1, the malware checks for tasks and downloads them over HTTP or DNS.

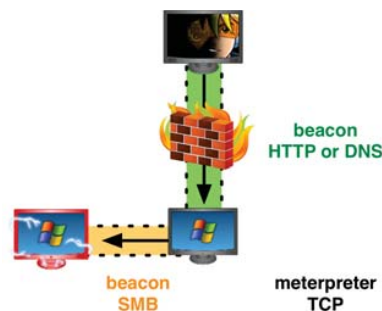


Fig. 1 Scheme of "Beacon" payload as designed by Cobalt Strike [17]

Later, it goes through these domains each time it has to use its beacon to signal back to C2. In case one domain fails or is blocked, the malware will go to sleep and wait until the next domain is available. Multiple domains or hosts use resilience to network communication defence activity.

According to Cobalt Strike, one of the most important malware functions is the specifically tuned *sleep function*, which specifies how long the malware has to wait between checks. The malware supports *asynchronous* and *interactive* communications. *Asynchronous* communication is low and slow, the malware will call back to the C2 server, download the task and then go to sleep. *Interactive* communication is

frequent and fast, the malware communicates to the C2 server in real time. These two types of communication methods are defined by the *sleep function*. If the sleep time is set to 0 seconds, then the malware will beacon back to the C2 server two times per second to maintain persistent communication. Otherwise, if the sleep time is to non-zero, like 20 seconds, then the malware will beacon every 20 seconds. What is more interesting, the malware can vary these intervals, using 20% jitter factor to vary the sleep time interval.

The above example only specifies one type of C2 server; in order to improve persistent ability, Cobalt Strike also provides distributed operation functionality, which means multiple servers are involved in distributed beaconing. In this case, it means multiple IP address. If a sub-group of C2 servers failed, it is still hard to block all the malicious activities that rely on other server. However, larger number of C2 servers increases the probability of detection, because too frequent connections may unveil the malicious footprint. This is also called *Low latency C2* due to its frequent beaconing. Thus the author of the malware will usually prefer to have a sleep time of no less than five minutes, while a longer sleep time is even better. This long beacon interval behaviour is called *High latency C2*, since longer beacon time leads to an inability to detection it. This can be explained by the sparsity of malware communication traffic in the context of a much higher number of benign communication activities.

C. DNS Logs as a Source for Compromise Indicators

DNS can be considered as one of the malware beaconing methods that is difficult to detect. The problem is related to the fact that DNS server functionality is a core of the Internet communication. The network communication is based on the FQDN (fully qualified domain name) to ease human memorizing and add a flexible level of topology abstraction. According to the Google DNS tutorial [18], one of the main DNS records is *A* that maps FQDN human-understandable address to physical IP addresses. OpenDNS made a study [15] in which DNS tunnelling was described as a contemporary technique. When malware requests a host, the infected machine sends a DNS query for an IP of C2 server to the internal/external DNS. By returning the DNS response, C2 can send a TXT record that may consist of 184 bytes of base63 encoded data as was studied by Farnham et al. at SANS [19]. Therefore, it can be a potential way of hiding malware job-scheduling. The advantage of this approach for an attacker is that there is an enormous amount of benign DNS records, which make it difficult to detect malware communication. Therefore, we can state that analysis of DNS logs need to be done with respect to periodic communications to detect possible targeted attacks and infected hosts. The following challenges can be named:

- 1) *The sleep time is not predictable.* There is no preliminary knowledge about the time when malware becomes active and sends beacons.
- 2) *Multiple period usage.* Attackers utilise one time interval for a period of time and then change to another time interval under some logical event or scheduling, or even use unique time intervals every time.

- 3) *Time variation.* Attackers take measures to prevent detection of the connection/communication between the compromised host and the C2 server, by varying the sleep time to make it appear as if it were a known-to-be-good communication. Command Five research made a study on sequences of communications for different protocols as presented in Fig. 2. We can see that experts have to look into the logs to be able to detect such variations manually.

PROTOCOL	TYPICAL BEACON INTERVAL* (SECONDS)
LURK	26
X-Shell C601	36
Update?	1 to 13, 12±3, 16, 104±3 or 200 ±15
Murcy	11
Oscar	12±2, 13, 15, 16, (55 or 155±5), (7.5, 8.5 or 15), (45, 55, 106)
BB	8
DB	4 to 92
Qdigit	60

* Commas indicate that the interval changed between victims. Brackets indicate that a variety of intervals were observed from a single computer.

Fig. 2 Common time intervals between communications used in different attacks [20]

- 4) *Noise.* The target host can sleep for a short period of time, while the malware can be asleep longer.
- 5) *Multiple channels usage.* A host may beacon to a single server for a limited time and then shift to another server.
- 6) *Benign beacon.* Some benign applications, like system updates or mailing clients, produce regular beacons.
- 7) *Needle in a haystack.* A large enterprise generates huge amount of logs. So, tracking of a relatively minute amount evidence within large scale DNS log files is like looking for a needle in a haystack and there are also time restrictions. IT security managers hope to detect malicious activity as fast as possible and once the malicious action has taken place, a warning should be generated.
- 8) *Near real time detection.* Beacon detection is not aimed at protection, yet identification of malicious beacon events as soon as possible reduces costs for enterprise. It might be hard to make it real time due to vast amount of network traffic. Also because the malware only broadcasts back to C2 servers *after* it has been launched, the deletion or theft of data on the targeted network has already begun. Hence, identifying malicious beacon events as soon as possible helps to reduce defence costs for targeted enterprise.

We can see that the majority of these challenges can be met by applying periodicity detection. However, many things need to be considered when analysing real DNS logs.

D. Periodicity Detection in Communication

Periodicity detection and mining can be used in different areas and represent a set of methods targeted at finding events that happen frequently with some deviations according to survey by Chitharanjan [21]. From the literature, we can see that there can be found a number of works that apply beacon detection in network traffic communication. Wang et al. [22]

proposed a circular autocorrelation based periodicity detection algorithm to detect repeating communications. However, they use a confidence value to evaluate detected events which will cause high false positive rate. Qiao et al. [23] proposed a method to detect P2P bots by mining the regional periodicity. They applied the PARPER algorithm, which is also an autocorrelation based algorithm, to identify the repeated events launched by malware C2. Their work is the first mention of a partial periodicity detection method that also handles noise. Van [13] also used an autocorrelation method to detect beacon events. Both methods examine all of the network traffic, but they are unable to identify individual events with high precision. The latest work by Parunak et al. [24] used a simple mathematical method to identify beacon events, but their method can only handle simple noise. However, their work is the first to analyse beacon sleep time and they hold the opinion that malware beacons can be differentiated from legitimate beacons through the sleep time. To the authors' knowledge, there has not been any work done on the application of symbolic periodicity detection for mining DNS logs with the intent to detect and understand malicious beaconing. Timestamps in DNS logs can also be considered as periodic events that can be linked to targeted attacks and malware.

III. PROPOSED METHODOLOGY FOR BEACONING AND TARGETED ATTACKS DETECTION

This section contains the insights into the proposed methodology, including beacon detection and attack capture modules. Our goal is to create methods for DNS log analysis and events correlation. We present the following goals that need to be achieved by our method:

- 1) *Low Latency.* We assume that the time interval of the beacon is short enough that the infected hosts will communicate with the C2 server several times (≥ 3) per day. Hence, we look into data that was generated over a whole one day to detect frequent beacon behaviour.
 - a) *Detection on Single Host.* Given any infected host, find the malicious domain which the host beacons to. Then, identify the other potential malicious domains involved in the attack.
 - b) *Detection on Entire Enterprise Network.* Try to detect attacks when no infected hosts are given on the selected date.
- 2) *High Latency.* We only consider malware communication at a fixed time every day, disregarding if it will communicate to the C2 server afterwards. Hence, we consider weekly data as a whole to detect high latency beacons.

No matter which C2 communication method is used (push or pull), the infected hosts will send feedback information to the C2 server. This will leave evidence on the DNS query record. Therefore we are only concerned with the A record type DNS log (A record DNS corresponds to IPv4, which is the IP address type most frequently utilized by attackers). It demonstrates queries from the internal host to the internal DNS server. However, the address queried from the internal DNS server may either point to the internal client or to a

client outside enterprise. Thus, we extract logs that query external client addresses through the internal DNS server. Our suggested methodology is represented in Fig. 3. The purpose is to present it as a framework, suitable for different enterprises.

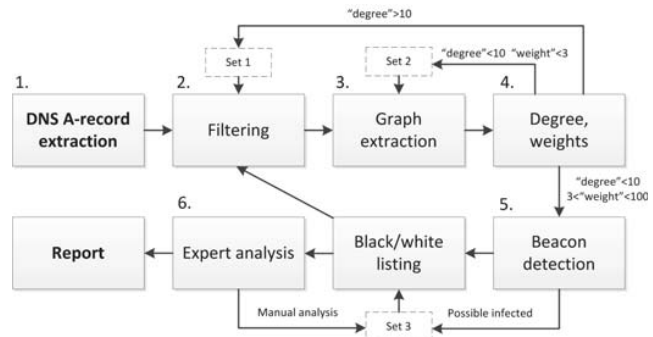


Fig. 3 Proposed iterative methodology of attacks detection from DNS A-records

The steps of the beacon detection in DNS logs are as follows:

- 1) A-records extract only IPv4 addresses from the DNS logs, since they are the most commonly used addresses. TXT records and their content are beyond our scope for now. The following 6 types of records are extracted: Query from internal machine to internal DNS server, from internal DNS server to internal DNS server, from internal DNS server to external DNS server with three corresponding responses.
- 2) The main task of data pre-processing is representing the filtered log data as an undirected graph, which provides an clear representation of the log file and communications between internal hosts and external domains. The graph's vertices represent host IP address and domain names, while each edge corresponds to one query from an internal host to an external machine. The attribute of the edge is the timestamp of the query. With this step, we intend to decrease the extracted log size. We introduce the "degree" attribute, which is assigned to each domain visited by internal hosts and records the number of individual hosts in the enterprise that visited this domain. We are only concerned by domains with a "degree" greater than 10. We think that domains visited by more than 10 hosts in an enterprise is a *benign domain*, while "degree" < 10 can be malicious. Even through there's a possibility that more than 10 hosts are infected in one targeted attack, we believe that IT departments have the ability to identify such large scale attack activities in an enterprise. The results are put into SET 1 as sketched in Fig. 4. So the "degree" > 10 threshold works as a white list and gets additional input from the Step 4.
- 3) Here we introduce another attribute, "weight", which denotes the number of times the host queries the specific domain. In this case, each domain has a "degree" and each individual host-domain pair has "weight" attributes. Then, we only take into consideration host-domain pairs with "weight" value greater than 3 and less than 100. The

domain	degree
zeffirelli.webb.bin	2
nosediving.webb.bin	2
bdi8vlwuhr	1
abnwb1.webb.bin	1
mo.abjpyljom5x.wad	1
economists.webb.bin	2
antitoxins.webb.bin	1
aj6nfavg70arz4v6laq4iod26...	1
bh46vyl793lvqw-qgemjnd0z-8...	1
necessitating.webb.bin	1

Fig. 4 Example of content in SET 1, where "degree" < 10

reason for this is that in real time processing mode, if a host query to one domain more than 100 times during a short time, we assume that such anomaly behaviour will obviously detected by security products employed in enterprise; for example, malware utilizing Domain Name Generation Algorithms (DGAs). Another concern is time consumption, since it takes more time to identify beacons from larger quantities of queries, we are planning to implement our method on a parallel processing platform. The results of this step are preprocessed data that are placed in SET 2 as shown in Fig. 5.

domain	host	weight
promissory.webb.bin	74.92.157.192	1
frequent.a.glazes.wad	74.92.208.83	1
a.v.promoting.wad.friend.noe	74.92.208.83	2
cot.adtn3q.wad	74.92.208.83	1
yolks.acp.val	74.92.29.150	2
cot.aap8-ry8dse5a.wad	74.92.254.164	2
cot.odin.sampler.don	74.92.49.40	2
footpaths.webb.bin	74.92.39.42	2
cot.aauw2-x-lwcl7a.wad	74.92.94.119	2
genial.intermarriages.wad	74.92.83.44	1
...

Fig. 5 Example of content in SET 2, where any of the domains were visited less than 10 times ("degree" < 10) and not connected to any hosts more than 3 times ("weight" < 10)

Then, an undirected graph is created to represent the resulting log records along with information from SET 2. It contains edges with the domains queried by its host 1, 2 or 3 times. We believe that a host, which exhibits beacon behaviour, should connect to the C2 server for at least 4 times. SET 2 is only used when we consider data as a stream and the content in this set will dynamically change over time. However, when we look the entire day's logs as a whole, we do not have to be concerned with SET 2 since we only need the corresponding edges whose weights are greater than 3. After filtering with SET 1, we represent our log data as an undirected graph which could help us to easily understand and analyse the log file. The graph's vertices represent host IP address and domain names, one edge corresponds to one query from an internal host to an external machine. The attribute of the edge is the query's timestamp. Figs. 6 and 7 represent mappings of internal to external hosts and graph representations, respectively.

- The beacons are detected based on the resulting data and represented by a graph.
- By using expert knowledge, SET 3 is composed from detected possible infected hosts. SET 3 is a list of beacons detected, each beacon is a host-domain pair, which is marked as malicious or benign.

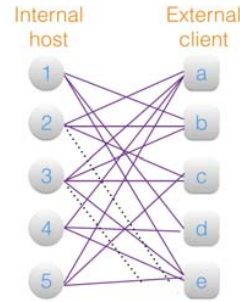


Fig. 6 One domain may be linked with one or multiple hosts



Fig. 7 Subgraph for a specific host

A. Beacon Detection Method

Here, we describe the analysis of preprocessed "suspicious" timestamps for beacon detection, as well as the choice of method, which includes four candidate periodicity detection algorithms. Fig. 8 presents a diagram of the method.

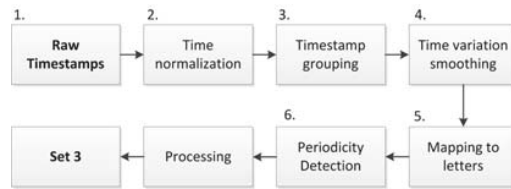


Fig. 8 Periodicity detection method

1) Proposed Suspicious Periodicity Detection Methodology:

The following steps are proposed:

- In order to check whether the time interval between two consecutive queries repeats over time, each timestamp is first transferred as the time elapsed from the beginning of the log collection, or the beginning of the period of interest, for the sake of maintaining a consistent the order of events.
- When it is convenient to gather all the timestamps for each pair host-domain, a sequence of query timestamps could be represented by a vector which has length equal to the number of queries minus one. For example, if a host "1" queries to external machine "A" for 14 times, then its time interval sequence will be represented by a vector with length 13.
- To prevent detection, an attacker may deliberately add time variations to the fixed sleep intervals, making the time interval vector resemble a randomly generated number. In order to tackle this problem, our idea is to treat all the numbers which are approximately equal to each other as

equal numbers; this effectively smooths the time variation within a certain scale. For instance, if 15 seconds time variation is allowed in the case of 2,400 seconds sleep intervals, then the entire time interval which is located between 2,385 to 2,415 maps to 2,400.

d) Each digital number vector corresponds to a letter vector in which different numbers are substituted by different letters, similar letters are represented by a random letter. A timestamp-letter transformation sequence is depicted in Fig. 9. Since the periodicity detection algorithm requires a different format of input: Letter or digital number, then each digital number vector corresponds to a letter vector in which different numbers are substituted by different letters and similar letters are represented by a random letter. For instance, (1,800, 1,800, 1,800, 3,600, 3600, 3,600) are denoted in letter by vector (a,a,a,b,b,b) as well.

Raw timestamps	1	2	3	4
2013-03-07 17:04:20.000001				
2013-03-07 17:13:17.000009	666,260	537	483	a
2013-03-07 17:25:25.000003	666,797	728	483	a
2013-03-07 17:36:15.999968	667,525	650	483	a
2013-03-07 17:44:18.000005	668,175	483	483	a
2013-03-07 18:05:12.000001	668,658	1254	1081	b
2013-03-07 18:05:12.000001	669,912	1320	1081	b
2013-03-07 18:27:12.000025	671,232	1081	1081	b
2013-03-07 18:45:13.999993	672,313			

Fig. 9 Timestamps transformation in beacon detection method

e) The periodicity detection algorithm is applied to detect whether there are suspicions domains that might be beaconing to external hosts.
 f) All infected domains are stored in the SET 3.

2) *Choices of Periodicity Detection Algorithms:* Four algorithms were selected based on the literature review for Beacon detection: Suffix Tree Based (STNR) [25], Dynamic time warping based (WARP) [26] and Convolution based (CONV) [27] by Mohamed G. Elfeky in 2005, Autocorrelation based (PARPER) [28] by Christos Berberdisin in 2002. Given a time sequence denoted by an alphabet vector, the sequence is called symbol periodic if individual symbols are repeated periodically. The sequence is called *segment periodicity* if we can divide the sequence into several segments with same length, and all of the segments are approximately or exactly the same. Another type of periodicity is called *partial periodicity*, if a pattern or a segment is only repeated periodically in a consecutive segment of the sequence. Note that in our case, a time sequence can be denoted by an alphabet vector. So, our tasks are to identify whether this sequence has periodic component sequences, what's the period and the repeated pattern, even if there's noise in the sequence.

B. Attacks Event Detection Module Based on Beaconing

After the malicious beacon has been identified, this step searches for other infected hosts and potential malicious domains related to the target attack. We classify a potential malicious domain as a "bridge domain" or a "rare domain", as mentioned earlier. The rules for detecting malicious domains and hosts are depicted in Fig. 10, (b) and as follows:

- 1) We assume that a host connected to a malicious domain has been compromised directly or indirectly by the attack. So all the hosts linked to domains which show malicious beacon behaviour are considered infected hosts.
- 2) Domains lying on the shortest path and only connected to the infected hosts are classified as "bridge domains". This is where the shortest path is defined as the path that relates two hosts through one domain vertex only.
- 3) There's a group of domains that are only queried by one potentially infected host and not visited by benign host before the beacon take place. We name those domains as "rare domains", which may be used to send malicious e-mails or download additional malware to bypass security checks before C2. So a large quantity of "rare domains" may be malicious or benign. The purpose of this step is to capture all malicious activity and to narrow the analysis scale as much as possible.

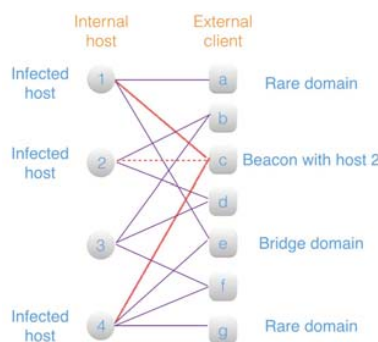


Fig. 10 Structure of beacon interconnections

Suppose that a beacon between host 2 and domain c is detected (the red dotted line), c is also connected to host 1 and 2 (red straight line); according to the rules, they are also potentially infected hosts. Domain e is linked to host 1 and 4 (except domain c which is a beacon domain); therefore, domain e is marked as a "bridge domain". Domains a and g are linked to hosts 1 and 4 respectively, and do not have links with other hosts; hence, we classify domains a and g as "rare domains". Domains b and f are also linked with benign host 3, even though they link with infected host. So, we think that such domains have less probability of being malicious.

IV. EXPERIMENTS

This section describes the experimental setup. We divide all experiments into two major parts: (1) Selection of the best periodicity detection algorithm for the described task, (2) Test of the beaconing analysis in DNS log files, to find out the best performing periodicity detection algorithm.

A. Experimental Design

To perform both sets of experiments we used the following hardware. Periodicity detection algorithms were implemented in Python and tested using MacBook Pro (4 cores CPU and 16 GB RAM memory). For the beacon detection (low latency and high latency), we utilized the VDS server available at

the Testimon Research Group. It had 3 cores (6 threads) of Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz, 32 GB of quad-channel RAM Kingston PC-1600, Ubuntu 14.04 64 bit installation on RAID SSD plus 2TB of HDD space for DNS logs pre-processing. For the second part of the experiments, we utilized the recently introduced framework GRAPHLAB developed by Low [29], which is a fast Machine Learning tool with parallel processing capabilities. This is an important tool for mining the connection between the internal addresses and external hosts.

B. Experimental Data

The data we used are DNS logs of the Los Alamos National Laboratory published in 2013¹. They are real logs from a large web-site but sanitised to conceal their origin and the actual referenced host names. Those logs contain name resolution requests from several simulated attacks, and include different stages of initial infection: Including initial callbacks, downloads of additional malware and C2 callbacks. When we worked on this paper, only 15 days of data (more than 400G) were available, each day containing a simulated attack. The IP address and domain name involved in the attack give label information. Days 09/3, 10/3, 14/3, 15/3, 17/3, 18/3, 19/3, 20/3, 21/3 have only one IP address that was labeled as an infected host. Days 07/3, 08/3, 11/3, 12/3, 13/3 have multiple infected IP addresses given. Then, day 22/03 has no infected host. For each of the 15 days, there are multiple malicious domain names documented according to the simulated attack. The task is to identify as much of the attack (infected hosts and malicious domains) as possible. We are only concerned with the A record DNS log, which corresponds to the most frequently used IP address type utilized by attackers: IPv4. It demonstrates queries from the internal host to the external client address through internal DNS server. The main idea of data pre-processing is to represent the filtered log data as a undirected graph. The graph's vertices represent host IP addresses and domain names. One edge corresponds to one query from the internal host to an external machine.

C. Reliability Test of Periodicity Detection Algorithms

According to the challenges mentioned in Section II, the periodicity detection algorithm should be capable of detecting not only a single period sequence, but also multiple period sequences that includes noise. More specifically, we should test the reliability of beacon detection algorithm with respect to following requirements:

- *Various types of periodicity.* There are three kinds of periodicity: *Symbol periodicity*, *Partial periodicity*, *Segment periodicity*. *Symbol periodicity* means only one symbol in the time sequence is periodic. For example, in $S = afgathargaoka$, only a repeats once every two symbols. *Partial periodicity* means that only a part of the sequence shows symbol periodicity. For example, we add random letters in front of and at the end of sequence S , giving us $S = GHIfgathargaokaQIO$, now S has

a partial periodicity with the repeat pattern a starting from position 3. *Segment periodicity* means the entire sequence only contains a repeated pattern; for example, $S = abcabcabcabc$, with repeat pattern abc which we call a period. *Segment periodicity* may include the following situations. (1) Some periods contain noise, while others are clear periods; so we call it perfect periodicity. For instance, in $S = abdabcabcabc$, the third position of the first period is noise; when we change it to c , the sequence will have perfect periodicity. However, if we consider the period as ab which is a regular expression, then the sequence S has perfect periodicity. (2) Another situation is when the length of the period is one, which means the sequence contains only one symbol, like $S = aaaaaaaaaa$, this is a normal beacon with fixed sleep time a and without time variation and noise.

- *Various length of period.* The length of the period could be longer than one symbol. However, in real cases the length of period is not predictable. Thus, the detection algorithm should be capable of automatic detection of various period lengths. For example, if the malware sets two beacon time intervals, the algorithm should identify the beacon time sequence, even if the time sequence is $S = ababababab$ or $S = aaaaaabbbbbbb$.
- *Various length of sequence.* The host-query-to-domain time intervals could vary in a range from one to thousands, which results in the length of time interval sequence ranging from zero to thousands. In our experiments, we only are concerned with the time interval sequences that have length greater than 3 and less than 100. The algorithm should be able to detect beacons of various sequence lengths and partial periodicity or segment periodicity, independently of the symbol periodicity.
- *Various kind of noises.* Noise complicates detection, more specifically; it will reduce the true positive rate. However, we need to control the detection rate even when the time interval sequence contains noise. This is always the case in real-world communications systems. In our time variation smooth module, noise will be generated artificially, based on a random noise process probability distribution. Hence, periodicity detection algorithms should have strong resilience to various types of noises, which including substitution, insertion, deletion, and even mixtures of periods.
- *Time performance.* Thereby, the time complexity of detection algorithms should be as low as possible for the sake of fast processing speed. A time interval sequence could be a random sequence, a periodic sequence or even a perfectly periodic sequence. Independently from this fact, the longer the sequence, the more time will be consumed. We implemented our experiments on a parallel processing platform which will assign sequences with different length and periodicity to different processors. Thereby the time complexity of detection algorithm should be as low as possible in order to achieve fastest possible processing time.

¹<ftp://ftp.lanl.gov/public/pflarr/>

D. Low Latency Beacon

In the low latency beacon detection experiments, we looked on the data as (1) a stream or as (2) a daily batch of data. Since each day contains only one simulated attack, it means that all of the documented attacks belong to low latency beacons (if they have beacon behaviour). After the graph creation for an entire day's log file, the experiments are arranged as follows:

- *Detection of on a single infected host.* Given any one of the infected hosts, detection of its beacon is an indicator of malicious activity. For each given IP address, we extract a subgraph that contains all the query logs that belong to this host. For days 09/3, 10/3, 14/3, 15/3, 17/3, 18/3, 19/3, 20/3, 21/3, which only contain 1 infected host, the subgraph contains vertices linked only to that host directly. For other days, where multiple infected hosts are documented, we choose any one host. Its subgraph contains vertices which link to the given host through less than four edges. It means that the distance between vertices and a given host is less than 4. Afterwards, for each domain vertex, we extract the processed time interval sequence as the input to the selected periodicity detection algorithm. If it shows beacon behaviour, an alert is raised. Suppose that the beacon has been detected, the next step is to use the detected beacon to find other infected hosts (if exists) and malicious domains.
- *Detection of attacks on entire enterprise network.* In this case, no preliminary knowledge is given, so we need to detect the beacon and understand the malicious activity. In this case, we will use all the data which documented attack information as training data, while the last day (22/3) is used as a test day and does not contain any information regarding the attacks.

E. High Latency Beacon

This part of the experiments are used to understand high latency beacons and test whether our methods are able to detect them. 15 day's data are divided into two group. The 1st group contains the first 9 days of log files, the 2nd one contains the last 6 day's data. The experiments are performed in two parts. (1) For those 9 days, we first use 7 days as training data, the 8th day as test data. Then, 8 days as training data to test the 9th day's data. (2) Further, the 9 days of data are used to train our module, while last 6 day data are used for testing. Similarly with the low latency beacon detection scenario, a set of allowed time variations will be tested on all the experiments.

V. ANALYSIS OF RESULTS

This section aims at testing whether the periodicity detection algorithms are able to handle the various beacon detection challenges mentioned earlier. Comparisons between four algorithms are presented and the best performing algorithm will be applied in our beacon detection method. First, we define *Confidence* in period detection as the evaluation criteria:

$$\text{Confidence} = \frac{\text{number of successful detections}}{\text{total number of detection}} \quad (1)$$

The value of the confidence will be in a range from 0.0 to 1.0. The higher the confidence value, the better the detection algorithm performs.

A. Reliability of the Periodicity Detection Algorithms

In this experiment, we test reliability with various lengths of period, which is set to a value in the range from 1 to 10. We generate this periodic sequence 11 times using a random uniform distribution. For instance, if the period is *AB* which has length 2, then the sequence will be *ABABABABABABABABABABAB*. The experiments were repeated 100 times, each time the length of period is fixed but the content of the period is different. At the end, we count the overall number of periods being detected and calculate the *Confidence* value.

Then, we perform noise resilience tests with five types of noise: *replacement, insertion, deletion, mixture of insertion and deletion, mixture of replacement, insertion and deletion*. The length of period is fixed to 5 and the elements of the period are generated randomly from a uniform distribution as well. Therefore, the total length of sequence is 50. The noised components are randomly generated and added to random positions. For example, if it is substitution noise and the randomly generated position is 23, then the 23rd element in the sequence will be replaced by another randomly generated element which is totally different from all the other elements in the sequence. The noise percentage increases gradually, until it reaches 50%, which means there will be 24 elements added to the original sequence. All experiments for resilience testing were implemented 100 times. Fig. 11 shows the results of these experiments.

Fig. 11 (a) indicates that all four algorithms are able to detect different period lengths with confidence to almost 1.0. Figs. 11 (b), (c) demonstrate the experiment results for noise resilience testing. The abscissa represents the percentage of noise added to the original sequence, the ordinate denotes the *Confidence*, different algorithms are represented by different markers. We can identify that STNR is the most reliable algorithm against various types of noise and also various types of periodicity.

B. Time Complexity Test

Three types of sequences are tested: *Perfect periodicity with one symbol, Perfect periodicity with multiple symbols in period and Random generated sequence*. The length of sequences varied from 0 to 200. For the 1st and the 2nd type of the sequences, the length was increased only a single time. For the 2nd type, a new period was added at the tail of the sequence each time.

Fig. 12 illustrates our test results. The abscissa represents the the length of sequence and the ordinate denotes the time consumed in seconds. The first type of sequence is perfect periodicity, which contains only one symbol and is depicted in the sub-figure (a). For example, the sequence $S = \text{AAAAAAAAAAAA}$. Second type of sequence in the sub-figure (b) represents perfect segment periodicity, for instance, $S = \text{ABCABCABCABC}$. The third sequence in the sub-figure (c) is

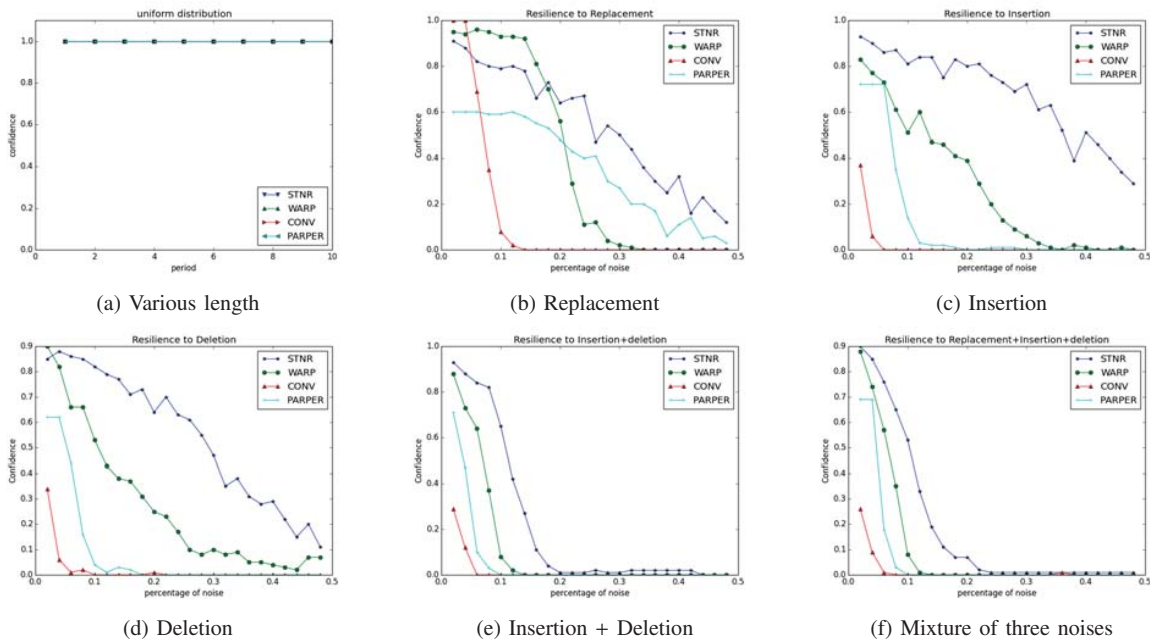


Fig. 11 Resilience to different types of noise

constructed with elements generated randomly from a uniform distribution, like $S = AJS DGAS GAKLDVOIER$.

Obviously, STNR and PARPER are the fastest algorithms against all three types of sequences. No matter what kind of sequence it is, processing time increased along with the increase in sequence length and a periodic sequence take less time to detect than a random sequence. Especially for STNR, when the length of sequence increases from 0 to 200, there's no significant difference in the processing time. To sum up, STNR performs much faster and is more stable than others based on this test. A comparison of complexity estimation for the four algorithms is given in Table I.

TABLE I
COMPUTATIONAL COMPLEXITY OF THE PERIODICITY DETECTION ALGORITHMS

Algorithm	STNR	CONV	WARP	PARPER
Periodicity	all type	segment	segment	partial
Complexity	$O(n^2)$	$O(n \cdot \log_2 n)$	$O(n^2)$	$O(n)$
Time performance	best	average	worse	good
Noise resilience	best	worse	worse	average

C. Low Latency Beacon Detection

Here, we present the results of low latency beacon detection based on the real DNS log data. We tested our method on 14 days' worth of data (excluding the 15th day, which is used to detect beacons on entire network). At the end, we detected beacons in 11 of the given days. In the other three days, we did not find out any beacons matching the documented attack, but we detect other beacons in the infected hosts. Table V-C lists the beacons we detected in each day. Column "Day" is the time when the log file was generated, "IP" column is the IP address of the infected host, "Number of Beacon"

records the total number of beacons detected corresponding to the infected hosts, the "Sleep time" column is the fixed time interval between each communication. If there is time variation detected in the time sequence, "yes" will be marked in time variation column; this indicates that the time gap between each two C2 varies slightly. The last column is the total number of queries that were recorded in the DNS log file.

Looking at the sleep time column, we find that attackers normally use only one fixed sleep interval for all infected hosts. An exception can be seen in day 11, when the attacker used slightly different sleep times for two infected hosts. For the time variations in attacks, we find 8 out of 15 days are using slight time variations. Since we set the allowed time variation to 2 seconds, all the time variations are located in range 0 to 2 seconds. However, it might be because an attacker set the time variation deliberately or it was caused by a delay in the package transfer between a router and a DNS server.

By designing an attack capture module into our method, we detected 27 potentially infected hosts in total, while the total number of documented infected hosts are 17. We also found that if there are multiple machines enrolled in the attack, like on days 07, 08, 11, 14, 15, 17, 21, then the "bridge domains" that we defined are able to cover almost all of the documented malicious domains. In addition, we identified 6 extra suspicious beacon behaviours, in addition to the documented attacks. The results are as in Table IV.

We also compared sleep time intervals used in each specific attack. Only day 8 utilizes another unique sleep time, 1,863 seconds instead of 1,703. While days 14, 15 and 21 are using the same sleep time as other C2 servers. Moreover, we tested our method on an entire enterprise network instead of a single host. The 14 day's data are considered as training data, and the last day's data as test data. We detected 5,421 unique hosts that

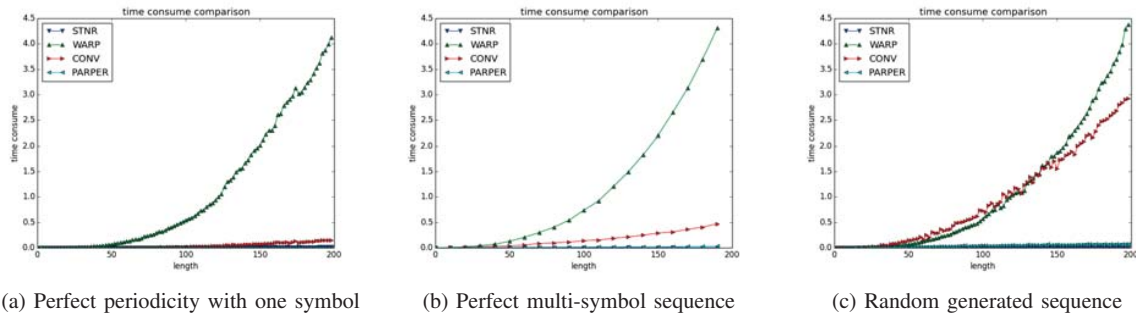


Fig. 12 Time performance test

TABLE II
MALICIOUS BEACON DETECTED WITH 2S TIME VARIATION

Day	IP	N ^o of Beacons	Malicious Domain	Sleep Time	Time Variation	N ^o of Query
07	74.92.150.58	1	aftible.noe	2,700	yes	13
07	74.92.11.100	5	aftible.noe	2,700	yes	10
07	74.92.229.13	1	aftible.noe	2,700	yes	9
08	74.92.196.178	1	lisped.val	1,703	yes	30
08	74.92.16.82	1	lisped.val	1,703	yes	26
08	74.92.65.93	1	lisped.val	1,703	yes	19
09	74.92.112.52	1	misorganisation.noe	600	no	29
10	74.92.83.97	1	glazes.inkly.k3	1,500	no	4
11	74.92.65.93	1	mine.starving.wad.f8	1,700	yes	10
11	74.92.20.216	1	mine.starving.wad.f8	1,704	yes	30
14	74.92.4.129	1	winterwolf.dx	600	no	24
15	74.92.120.47	1	orbitals.ok	7,200	no	13
17	74.92.65.174	1	blerf.nc	1,783	no	9
19	74.92.144.170	1	rainbow-.c3	603	no	14
20	74.92.255.55	1	xu-l8jwxcetd.wad	1,100	no	13
21	74.92.159.71	1	carcassonne.b-	1200	no	17
Total	16	20	11	11	-	-

TABLE III
RESULTS FOR ENTIRE EVENT CAPTURE

Day	Infected host	"Rare domains"	"Bridge domains"
07	74.92.150.58	124	2(2/3)
	74.92.229.13		
	74.92.11.100		
08	74.92.196.178	476	20(3/3)
	74.92.65.93		
	74.92.64.222		
09	74.92.112.52	174	0(0/3)
	74.92.83.97		
10	74.92.65.93	67	0(0/3)
	74.92.65.93		
11	74.92.20.216	48	3(3/3)
	74.92.159.71		
	74.92.27.83		
14	74.92.4.129	362	4(3/4)
	74.92.16.82		
	74.92.120.47		
15	74.92.175.4	142	9(3/3)
	74.92.65.174		
	74.92.65.177		
17	74.92.144.170	2(2/3)	1(1/3)
	74.92.65.177		
19	74.92.144.170	3(2/4)	0(0/4)
	74.92.255.35		
20	74.92.255.35	5(2/4)	0(0/4)
	74.92.255.35		
21	74.92.159.71	665	2(1/4)
	74.92.88.72		

beaconed to 40,377 unique domains in the training data, and 326 unique hosts beaconed to 1,649 unique domains in the test data. This is quite a large number of beacons, which means that beacons are very common behaviour in the network. Almost

every host has installed some application that shows beacon behaviour. This creates a challenge in distinguishing malicious and benign beacon.

In all previous experiments of low latency C2 detection, we set allowed time variation to 2 seconds. Also there is a possibility that an attacker sets higher time variations to escape detection. Hence, we allowed time variations up to 10 seconds. Then, we detected 23 new beacons over all infected hosts. Through we have no idea whether those domains are malicious or not, it provides clues for identifying malicious beacons within small scale data.

Further, we analysed the time intervals used in detected beacons. Fig. 13 (a) gives a distribution of sleep times which we detected, most of the sleep times are found in the range 2-1,000 and around 4,000 seconds. While Fig. 13 (b) gives statistics for concrete sleep times, the most common beacon sleep time is 2 seconds, which means that most applications frequently call back to their server. Other less popular sleep times are 3,600 seconds, 1,800 seconds, 7,200 seconds, and 900 seconds. This demonstrates that most benign applications prefer to use Integer multiplier of 5 minutes or 30 minutes doe call backs to their server.

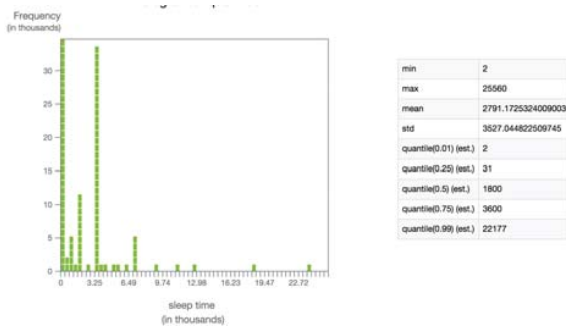
We compared the most common sleep time intervals used in benign beacons and malicious beacons detected in the dataset, as presented in Fig. 14. Most of malware sleep time intervals are contained in the benign interval.

TABLE IV
EXTRA INFECTED HOSTS DETECTED

Day	IP	Malicious Domain	Sleep Time	Time Variation	Noise	Nº of Query
08	74.92.64.222	lisped.val	1,863	no	yes	47
14	74.92.16.82	winterwolf.dx	600	no	no	57
14	74.92.27.83	winterwolf.dx	600	no	yes	55
15	74.92.175.4	orfitals.ok	7,200	no	no	6
15	74.92.244.3	orfitals.ok	7,200	no	no	7
21	74.92.88.72	carcassonne.b-	1,200	no	no	15
Total	6	4	4	-	-	-

TABLE V
EXTRA BEACON DETECTED WITH TIME VARIATION TO 10s

IP	Domain	Time interval sequence
74.92.120.47	cot.aac1axygu9177nm7u.noe	[3636, 3607, 3603, 3600, 3602, 3602, 3603, 4805, 3643]
74.92.83.97	aaqz7z-69qgdvzj.jeans.wad	[606, 623, 605, 920, 606, 610, 621, 603 ... 614, ..., 605]
74.92.83.97	ac3ak6imsj62a.bernard.crus...	[3639, 7260, 3606, 3744, 3691, 3636, 3609, 3666, 3664,...]



(a) Distribution of sleep time

Value	Count	Percent
2	23241	22.082969100376268%
3600	22376	21.261069514661666%
1800	11026	10.476606742427123%
3615	7117	6.7623807532971%
7200	5149	4.892440424157197%
900	2927	2.7811561704230168%
30	2277	2.1635437649652234%
1200	1530	1.453764585154498%
5400	1527	1.4509140663600775%
300	1456	1.3834517882254571%
60	817	0.7762912850138726%
10800	655	0.622363270115161%

(b) Frequency of sleep time

Fig. 13 Histogram frequencies of detected sleep times for low latency beacons

We have found that attackers use sleep times similar to benign applications, like 600 seconds, 1,200 seconds and 7,200 seconds. Some of the sleep times used in malware are identical to benign applications, which means that we cannot distinguish malware through beacon sleep time only. However, many benign applications prefer to use a common sleep time, like 5 minutes, 30 minutes, 60 minutes or 120 minutes. On the other hand, malware prefer to use some rare and uncommon sleep time, even though we cannot identify malware through its sleep time only. So, unusual sleep times may indicate a malicious beacon.

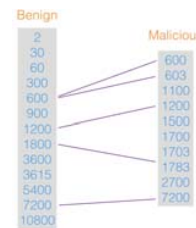


Fig. 14 Sleep time intervals found for malicious and benign beacons, in seconds

D. High Latency Beacon Detection

The last experiment was targeted on sleep time analysis in high latency beacons. Days 8 to 14 were set as the first week of data and days 9 to 15 as the second week. Allowed time variations were limited to 60 seconds. Only the first connection of each day is considered, because the first call back to server for some malware (or benign application) may be at a fixed time and afterwards creates call backs randomly. The results are as Fig. 15 shows. For example, some applications may check updates at a predefined time or call back to their server once the host machine starts. However, some malware may imitate those benign applications to hide their location. We thought that if some new application or software called back to one server at almost the same time every day, then this application may be malware installed on the host, especially for those hosts which do not update or install new applications frequently, like a live server or a commonly used workstation.

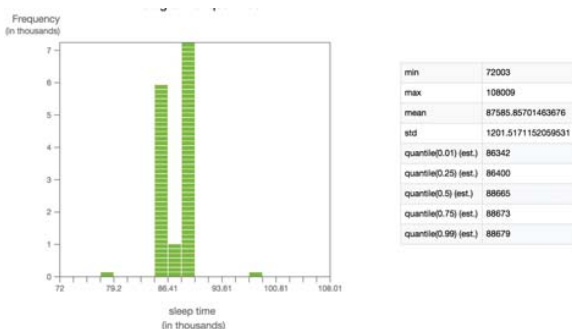
E. Overall Performance of the Proposed Method

We implemented all experiments on the parallel processing platform GraphLab, to be able to achieve fast processing speed. The speed is almost always consistent (around 112,000 records), using on our hardware setup mentioned earlier, regardless of the total number of events processed as given in Table VI.

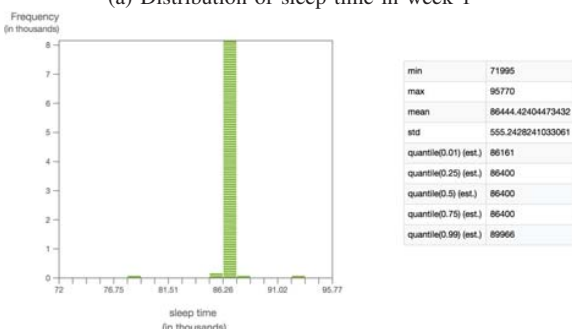
It can be seen that, on average, our method is able to process 114,957 queries which contains 9,061 events per second enrolled in beaconing. The average processing speed for each day in DNS logs is shown in Fig. 16. We think that if

TABLE VI
NUMBER OF DNS LOG EVENTS AND TIME SPENT FOR PROCESSING PER EACH DAY

Day	№ of DNS records	№ of beacon events	Required time, sec
07	111,737,320	9,482,854	964
08	90,618,325	8,489,702	731
09	58,906,031	7,732,506	535
10	49,992,677	4,964,274	442
11	108,203,056	7,235,972	908
12	105,792,961	6,177,169	875
13	104,077,438	5,051,064	841
14	104,536,592	6,482,604	886
15	78,931,777	15,793,766	686
16	52,387,778	7,414,774	509
17	89,380,486	5,452,700	797
18	94,515,370	7,913,983	871
19	92,526,276	8,008,275	845
20	97,388,332	6,077,968	864
21	74,399,167	6,255,342	678



(a) Distribution of sleep time in week 1



(b) Distribution of sleep time in week 2

Fig. 15 Distribution of sleep time for high latency beacon

we use CPUs with more threads, like Xeon Phi, the speed will increase further. But still there is a limitation on the number of operations that can be run in parallel and network bandwidth. We can state that such processing speed is relative fast and might be considered as real-time performance.

VI. CONCLUSIONS

Our work reviewed the state of the art in the beacon behaviour analysis for attacks detection in DNS log files. Previously, only a few papers focused on beacon detection and almost all of them did not pay attention to time variations in beaconing. In this work, tactics used in beaconing and beacon detection challenges were summarised and compared

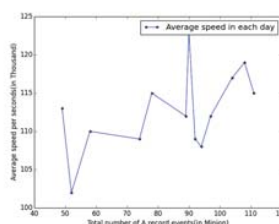


Fig. 16 Comparison of average processing speed for each day

with various periodicity detection algorithms such as the autocorrelation-based algorithms used in previous works.

We presented a malware beaconing detection method which has not been presented before, to authors' knowledge. The experiment results demonstrate that a suffix tree-based algorithm is the most reliable and fastest for beacon detection and satisfy all specified requirements. Our method takes time variation into account. Test results indicate that the method can reliably handle large-scale time variations. Previous works failed to detect 6 day's beacon out of 15, while our work detected more beacons, which could potentially originate from infected hosts. These are the contributions by our attack capture module. In addition, our experiments presented a different view on differentiation between malicious beacons and benign beacons, through sleep time analysis. It was shown that an attacker may use the same sleep time intervals as legitimate applications, yet an uncommon sleep time may indicate the existence of malware. One of the future works might be to create a bigger dataset which contains more attacks and cover more tactics in beaconing.

Another contribution of our work is utilization of parallel processing implementation using GraphLab library on a VDS server. It is capable of near real-time DNS log processing. The previous works did not pay much attention to this issue and we think that developing fast and efficient beacon detection mechanisms based on Machine Learning will benefit enterprise security.

ACKNOWLEDGMENT

The authors would like to acknowledge the valuable feedback provided by Dr. Carl Stuart Leichter. Also, the

sponsorship and support from COINS Research School of Computer and Information Security.

REFERENCES

- [1] "Targeted Attacks - Definition - Trend Micro USA." <http://www.trendmicro.com/vinfo/us/security/definition/targeted-attacks>. Access date: 2015-02-27.
- [2] "New anti-APT tools are no silver bullets: An independent test of APT attack detection appliances | MRG Effitas Blog." <https://blog.mrg-effitas.com/>. Access date: 2015-05-24.
- [3] "Compromise assessment," tech. rep., Mandiant, https://dl.mandiant.com/EE/assets/DS_CompromiseAssessments_140207.pdf. accessed: 08.12.2015.
- [4] "APT INFECTION DISCOVERY USING DNS DATA[info:lanl-repo/lareport/LA-UR-13-23109]." <http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-13-23109>, 2013. Access date: 2015-05-24.
- [5] N. Villeneuve and J. Bennett, "Detecting apt activity with network traffic analysis," *Trend Micro Incorporated*, 2012.
- [6] "contagio: Mandiant APT1 samples categorized by malware families." <http://contagiodump.blogspot.no/2013/03/mandiant-apt1-samples-categorized-by.html>. Access date: 2015-05-24.
- [7] "Command and Control in Fifth Domain COMMAND FIVE PTY LTD - Engineering Innovation | Research." <https://www.commandfive.com/research.html>, 2011. Access date: 2015-05-19.
- [8] Y. Low, J. Gonzalez, A. Kyröla, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new framework for parallel machine learning," *CoRR*, vol. abs/1006.4990, 2010.
- [9] X. Huang, "Understanding beacon for identifying targeted attack by mining large-scale log data," Master's thesis, Gjøvik University College, 2015.
- [10] A. Oprea, Z. Li, T.-F. Yen, S. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," *arXiv preprint arXiv:1411.5005*, 2014.
- [11] "NTP, Network Time Protocol." <http://support.ntp.org/bin/view/Main/WebHome>. Access date: 2015-05-19.
- [12] "How to use RSS feeds | Digital Trends." <http://www.digitaltrends.com/computing/how-to-use-rss/>. Access date: 2015-05-19.
- [13] L. van Duijn, "Research project-report beacon detection in pcap files," 2014.
- [14] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.
- [15] "The role of dns in botnet command & control," tech. rep., OpenDNS, http://info.opendns.com/rs/opendns/images/OpenDNS_SecurityWhitepaper-DNSRoleInBotnets.pdf, 2012.
- [16] A. P. T. S. C. Strike. <http://www.advancedpentest.com/>. accessed: 10.12.2015.
- [17] "Stealthy peer-to-peer c&c over smb pipes." <http://blog.cobaltstrike.com/2013/12/06/stealthy-peer-to-peer-cc-over-smb-pipes/>, December 2013. accessed: 10.12.2015.
- [18] Google, "Dns basics." <https://support.google.com/a/answer/48090?hl=en>. accessed: 12.12.2015.
- [19] G. Farnham and A. Atlasis, "Sans: Detecting dns tunneling." <https://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152>, February 2013. accessed: 08.12.2015.
- [20] C. F. P. Ltd, "Command and control in the fifth domain." https://www.commandfive.com/papers/C5_APT_C2InTheFifthDomain.pdf, February 2012. accessed: 11.09.2015.
- [21] K. Chitharanjan, "Periodicity detection algorithms in time series databases-a survey," *International Journal of Computer Science & Engineering Technology*, 2013.
- [22] B. Wang, Z. Li, D. Li, F. Liu, and H. Chen, "Modeling connections behavior for web-based bots detection," in *e-Business and Information System Security (EBISS), 2010 2nd International Conference on*, pp. 1-4, IEEE, 2010.
- [23] Y. Qiao, Y.-x. Yang, J. He, C. Tang, and Y.-z. Zeng, "Detecting p2p bots by mining the regional periodicity," *Journal of Zhejiang University SCIENCE C*, vol. 14, no. 9, pp. 682-700, 2013.
- [24] H. V. D. Parunak, A. Nickels, and R. Frederiksen, "An agent-based framework for dynamical understanding of dns events," 2014.
- [25] F. Rasheed and R. Alhaji, "Stnr: A suffix tree based noise resilient algorithm for periodicity detection in time series databases," *Applied Intelligence*, vol. 32, no. 3, pp. 267-278, 2010.
- [26] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Warp: time warping for periodicity detection," in *Data Mining, Fifth IEEE International Conference on*, pp. 8-pp, IEEE, 2005.
- [27] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Periodicity detection in time series databases," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 7, pp. 875-887, 2005.
- [28] C. Berberidis, W. G. Aref, M. Atallah, I. Vlahavas, A. K. Elmagarmid, et al., "Multiple and partial periodicity mining in time series databases," in *ECAI*, vol. 2, pp. 370-374, 2002.
- [29] Y. Low, J. E. Gonzalez, A. Kyröla, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.