2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

BAYWATCH: Robust Beaconing Detection to Identify Infected Hosts in Large-Scale Enterprise Networks

Xin Hu[*§], Jiyong Jang[†§], Marc Ph. Stoecklin[†], Ting Wang[‡], Douglas L. Schales[†], Dhilung Kirat[†], and Josyula R. Rao[†]

[†]*IBM Research, {jjang, mpstoeck, schales, dkirat, jrrao}@us.ibm.com*
[*]*Pinterest, huxinsmail@gmail.com*
[‡]*Lehigh University, ting@cse.lehigh.edu*

***Abstract*—Sophisticated cyber security threats, such as advanced persistent threats, rely on infecting end points within a targeted security domain and embedding malware. Typically, such malware periodically reaches out to the command and control infrastructures controlled by adversaries. Such callback behavior, called beaconing, is challenging to detect as (a) detection requires long-term temporal analysis of communication patterns at several levels of granularity, (b) malware authors employ various strategies to hide beaconing behavior, and (c) it is also employed by legitimate applications (such as updates checks). In this paper, we develop a comprehensive methodology to identify stealthy beaconing behavior from network traffic observations. We use an 8-step filtering approach to iteratively refine and eliminate legitimate beaconing traffic and pinpoint malicious beaconing cases for in-depth investigation and takedown. We provide a systematic evaluation of our core beaconing detection algorithm and conduct a large-scale evaluation of web proxy data (more than 30 billion events) collected over a 5-month period at a corporate network comprising over 130,000 end-user devices. Our findings indicate that our approach reliably exposes malicious beaconing behavior, which may be overlooked by traditional security mechanisms.**

I. Introduction

Sophisticated cyber security attacks and threats, such as Advanced Persistent Threats (APTs) and targeted attacks, are amongst the most advanced cyber security threats to date. The anatomy of such attacks can be described by three main phases (cf. Fig. 1): (a) the *initial compromise* of one or more devices within a target infrastructure, (b) the establishment of a *communication channel* enabling the threat actor to remotely control devices within the target infrastructure, and (c) the *execution* of the mission of the attack, such as access to and exfiltration of sensitive data or disturbance of processes. From a defender's perspective, many recent threat reports about high-profile attacks, campaigns, and data breaches (e.g., BlackPOS [17], Regin [23], Epic Turla [15, 16], Sony Pictures [27], APT1 [18], Duqu [4], or RSA SecurID compromise [21]) indicate that in most of these operations the initial compromise was extremely difficult to detect or prevent. For example, the threat actors leveraged sophisticated social engineering methods (to the level of dedicated language specialists [18]) or exploited zero-day vulnerabilities (e.g., via drive-by downloads, waterholing attacks) in order to place malware.

A common pattern in all these operations is that after the initial infection, the adversary controls the infected machines by establishing a stealthy communication channel between

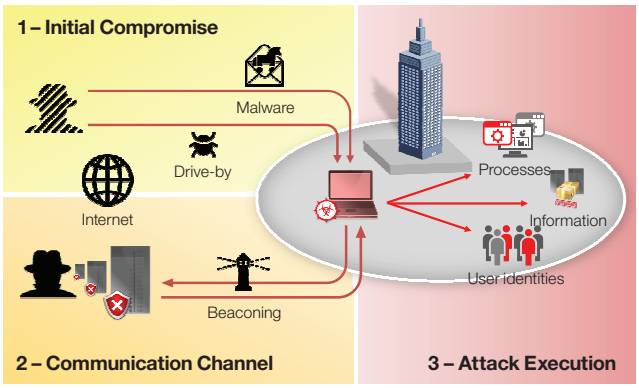

Fig. 1. Anatomy of advanced cyber security attack

the malware process within the target infrastructure and the adversary's command and control (C&C) infrastructure that is outside the perimeter. However, to the defender's advantage, the need for such interaction requires the exchange of network communication, and thus provides an opportunity to the defender to detect its presence. Due to security mechanisms (e.g., firewalls, IDS/IPS, NAT), which prevent an outsider from initiating a connection to a machine inside a security perimeter, such channels have to be initiated from inside. Typically, the attacker designs the malware to reach out to the C&C infrastructure on a regular basis to announce and confirm its presence and get further instructions. This behavior is referred to as "*beaconing*". Depending on the sophistication of the attack, not every infected device within an organization beacons to the C&C infrastructure; only the systems used as stepping stones [18], or a dedicated "proxy" machine communicate with the outside [23]. Another well-known form of beaconing behavior is observed in botnet C&C usage [6, 22], where up to hundreds of thousands of infected hosts announce themselves regularly and establish stealthy channels in order to receive instructions from their bot master.

Beaconing behavior can be characterized as a regular (sometimes highly periodic and sometimes not) sequence of requests or *beacons*. A beacon is almost like a heartbeat and consists of a short communication between the infected machine and one or more destinations on the outside. The frequency of such beacons depends on the attacker's strategy, which can be slow and stealthy or fast and aggressive. For example, in the wild, we observed beaconing attempts every 2–3 seconds as well as every 2 hours or even longer. In general, attackers use regular beacons to maintain visibility and control over the infected machines. HTTP(S) is commonly used as an

§ Both authors contributed equally to this work.
* X. Hu & T. Wang were with IBM Research while carrying out this work.

978-1-4673-8891-7/16 $31.00 © 2016 IEEE
DOI 10.1109/DSN.2016.50

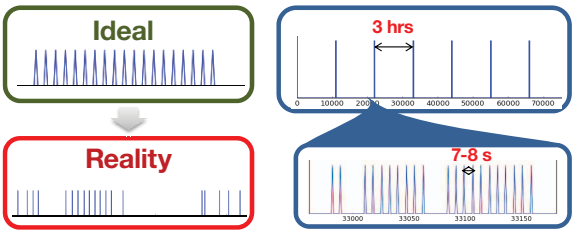

Fig. 2. Challenges: Expected patterns vs. real-world activity (left), multiple periodicities (right).

application protocol due to its nearly universal ability to cross security controls at network perimeter boundaries.

Detection of beaconing behavior is, however, not a trivial task but requires that the following four challenges are properly addressed:

Challenge 1: Beaconing is not an isolated event, but a sequence of temporally related events that have to be culled and analyzed in context. As a consequence, detection of beaconing is a "big data problem" as the traffic behavior of all communication pairs[1] over the network perimeter needs to be analyzed over an extended period of time for indications of beaconing behavior. In a large corporate enterprise network consisting of dozens of sites, it is easy to observe an average of 53 million distinct communication pairs every day using the HTTP(S) protocols only.

Challenge 2: While beaconing is characterized as a periodic communication in a regular interval, the ground reality is vastly different. There can be random gaps in the collected observations either because of devices going off-line or beyond the purview of observation, network down times, unanticipated issues in the data collection process as well as noise in the communication channel (left side of Figure 2) for a real-world activity trace collected at the network level from a device infected by the TDSS botnet [20]). The same device may reconnect from different IP addresses and the destination entity can have multiple IP addresses, making it difficult to track the context of the communication pair for constructing beaconing sequences.

Challenge 3: The nature of the beaconing behavior is completely at the discretion of the attacker, e.g., in terms of the choice of frequency and strategy employed. For example, the malware may omit a beacon or inject additional random beacons to reduce the predictability of the behavior. The right side of Figure 2 depicts the real-world activity network trace of a Conficker [6] bot beaconing to its C&C infrastructure. The malware sends beacons every 7–8 seconds for about two minutes, before it is dormant for about three hours, and then it repeats the same high frequency beaconing.

Challenge 4: Regular beaconing behavior does not necessarily indicate malicious activity: many legitimate applications exhibit network behaviors that resemble beaconing, such as regular update checks verifying software versions or anti-virus signatures, license checks, and e-mail or news polling.

Traditional security mechanisms such as firewalls, IDS, and web proxies attempting to match signatures are not able to cope with these challenges and keep the long-term history needed, and thus suffer from a high false negative rate.

In this paper, we address the problem of detecting large-scale beaconing behavior indicating malicious activities based on network monitoring. We introduce BAYWATCH, a comprehensive filtering methodology to analyze network interactions for identifying beaconing behaviors and pinpointing assumed infected devices and malicious communication channels. At the heart of BAYWATCH, we have devised a periodicity detection algorithm based on signal processing that is robust to noise, interrupts, and outages; the algorithm does not rely on strict periodicity but is able to capture regularity at different time scales. As an output, BAYWATCH provides a prioritized list of beaconing cases. We systematically evaluate BAYWATCH both on synthetic data and real-world data sets: two HTTP(S) traces (one of which is 5 months long, 30 B events) collected at the perimeter gateways of a major corporate network. In our evaluation and experience, about 26 suspicious beaconing cases were reported every day out of which over 96% of the top-ranked cases could be confirmed as truly malicious.

The key contributions of this paper are as follows:

- An 8-step filtering methodology (called BAYWATCH) to reliably identify beaconing behavior and separate legitimate from malicious beaconing—one of the first indicators of malware infection—consisting of a novel periodicity detection algorithm which is robust to real-world perturbations as well as pre- and post-filter operations.
- A highly scalable implementation of the methodology on top of the parallel and distributed Map/Reduce programming model [7] allowing iterative analysis of long time windows (up to months) for beaconing behavior.
- A systematic evaluation of the properties of the core beaconing detection algorithm using synthetic data sets as well as a comprehensive evaluation of the entire methodology by means of real-world data sets, which indicates that the output of BAYWATCH provides a manageable number of cases every day, in which 96% could be confirmed as truly malicious.

II. OVERVIEW AND ARCHITECTURE

In order to address the four challenges introduced in the previous section, we have devised a methodology based on eight filtering steps. The procedure departs with the assumption that every event in a given time span (up to weeks or months) may be relevant and needs to be taken into account and tested against the filters. Initially, the network events are grouped together by means of *communication pair* configurations. A communication pair is defined as a pair of source and destination endpoints, where each endpoint is represented by one or more of its features as listed in Table I.

The overall architecture of the filtering approach in BAYWATCH is depicted in Fig. 3. It consists of four main phases: (a) whitelist analysis, (b) time series analysis, (c) suspicious indication analysis, and (d) manual verification and investigation. At each phase, one or more filters are applied to the events. Figure 3 shows the data flow and the alignment of the filters.

[1] cf. Section II for the definition of a communication pair

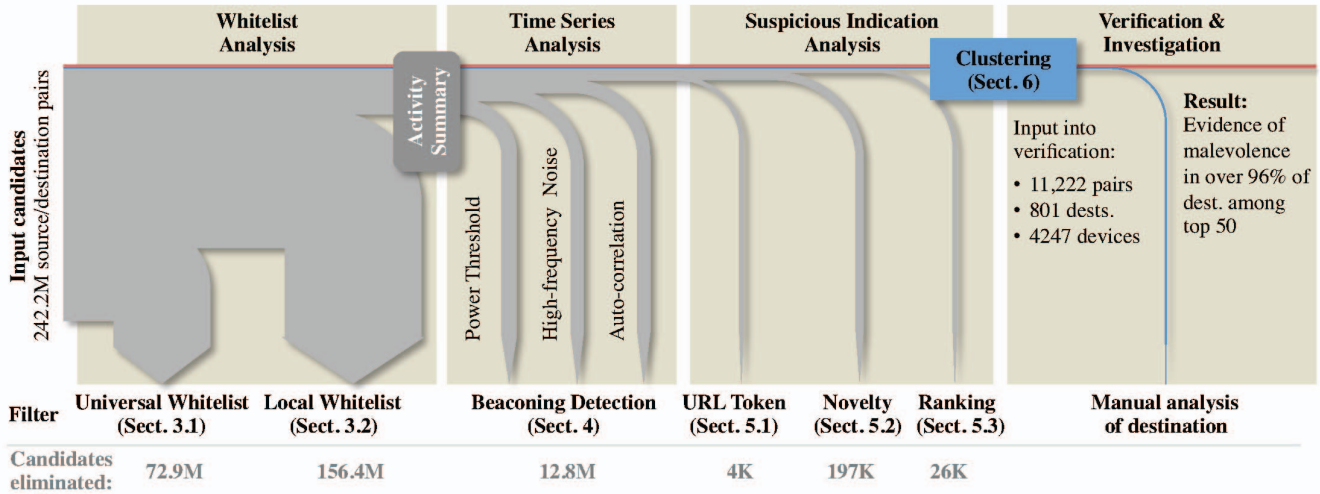

Fig. 3. Overview of the methodology and filtering approach of BAYWATCH. The numbers reflect a representative analysis of one month (January 2015) in the Web Proxy Log dataset.

The arrows pointing downwards indicate the (proportional) number of communication pairs filtered out (gray numbers in the bottom row) in a representative analysis over 1 month's worth of web proxy logs in our data set (cf. Sect. VIII for the description of the data set).

In summary, the four phases consist of:

Whitelist Analysis (Sect. III) Two whitelisting filters identifying legitimate and trusted destination features, inferred from global (i.e., universally applicable) and local (i.e., environment-specific) dynamic lists.

Time Series Analysis (Sect. IV) From non-whitelisted events, so-called activity summary (a compressed version of time series) is constructed for every communication pair that serves as an input to the frequency analysis and the periodicity detection algorithm. All communication pairs that exhibit no regular patterns are removed from the set of candidates.

Suspicious Indicator Analysis (Sect. V) All remaining candidate pairs are now analyzed along with domain-specific information for indications as to maliciousness to weed out legitimate beaconing behaviors and identify new types of beaconing. Candidates that are not eliminated are rank-ordered by means of multiple information criteria, and grouped by similarity using a machine learning based classifier.

Investigation & Verification (Sect. VI) The results are then presented to a security analyst for deeper analysis, investigation, and verification. Given that this is a manual step, it is preferred to minimize the number of individual cases to be investigated without sacrificing false positive/negative rates. We propose a bootstrapping process: using a small number of candidate cases and their investigation results as "training set," we train a classifier and apply it to the remaining cases.

III. WHITELIST ANALYSIS

The first phase of BAYWATCH consists of two filters to eliminate known beaconing and trusted communication pairs from the set of candidates. For this purpose, two whitelisting mechanisms are employed: (a) universal whitelisting, and (b) local whitelisting. This is important as it critically influences the workload of the time series analysis at the subsequent phase, which is data- and computation-intensive; hence, early reduction of the workload is beneficial for the overall performance. The intuition behind leveraging both whitelists is that popular destinations are not likely candidates for malicious beaconing (unless a majority of devices are compromised).

TABLE I. SOURCE AND DESTINATION PAIR FEATURES.

Source features	Destination features
IP address	IP address
Port number	Port number
Source MAC address	Full domain name
Device identifier	Top-level domain name
User Agent	AS number
	URL

A. Universal Whitelisting

The first filtering step builds on top of a universal whitelist that applies to every environment and comprises trusted or known destinations such as search engines, software update checks, web mail/e-mail polling, or news feeds (e.g., RSS feeds). In this context, a destination may be an IP address or a domain name. This whitelist is dynamic and can be extended with new destinations by means of update feeds.

As illustrated in Figure 3, using a public list of popular domain names as a whitelist, such as the top 1 million domain names reported by Alexa.com [1], proves useful by significantly reducing the computational need. One caveat is that this approach will fail to capture beaconing events to infected popular domains or covert channel-based beacons through public webservices. However, malicious usages of popular domains are more likely to be detected by the security community at large and exposed publicly [2]. Such threat intelligence can be promptly integrated into BAYWATCH via blacklists. When enough computational resources are available,

one can also simply disable this filter to perform more thorough beacon detection in the network.

B. Local Whitelisting

In addition to the universal whitelist, we also measure popularity of a destination as a ratio of the number of hosts accessing it and the total number of observed hosts. Destinations whose popularity exceeds a given (conservative) threshold τ_P are considered locally popular in a specific environment, and are whitelisted. This local whitelist removes candidates that are used by a subset of the population in a given environment (such as a specific anti-virus update server only used in the environment), which are not eliminated by the universal whitelist.

IV. Time Series Analysis

A beaconing behavior can be loosely defined as repeating activities at regular time intervals. It is challenging to mine periodic patterns from massive and noisy data. In this section, we will present the design of a multi-stage periodicity detection algorithm that leverages statistical features from both the time domain and the frequency domain, to eliminate noises and potential false positives, and to identify the most feasible candidate periodicities in the input data.

A. Algorithm Overview

Let $T = \{t_1, t_2, \cdots\}$ be the timestamps of connections between a communication pair. T can be transformed into an integer sequence $x(n) = \{x_0, x_1, x_2, x_3, \cdots, x_{N-1}\}$ where $n = 0, 1, ..., N-1$, and N is the length of the discrete time series. $x_i > 0$ means the source has made x_i number of connections to the destination at time interval i, and $x_i = 0$ means otherwise. Notice that the length of the time intervals depends on the granularity or quantization (i.e., sampling rate) of the time series, e.g., second, minute, hour, or day. To detect periodicity in $x(n)$, we adopt two complementary methods proposed in [25], i.e., Fourier transformation and circular autocorrelation.

Specifically, the time series analysis algorithm consists of three steps in sequence: (a) discovery of the most likely periodic components, (b) statistical pruning of high frequency noise, and (c) verification to retain only the most feasible candidate periods.

B. Step 1: Candidate Discovery

The first step is to discover periodicity candidates from the time series. To this end, we apply the Discrete Fourier Transformation (DFT) and convert the sequence $x(n)$ from the time domain to its corresponding frequency domain. Mathematically, the DFT converts a finite sequence of N numbers in $x(n)$ into a sequence of complex numbers $X(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-2\pi i k n/N}$, where $k = 0, 1, ..., N-1$, and the original signal is represented as a linear combination of the complex sinusoids $s_k = e^{i2\pi k n/N}/\sqrt{N}$ each of whose frequency is $f_k = k/N$. Essentially, the Fourier transform projects the original signal $x(n)$ onto these sinusoids and records their amplitude and phase with the coefficients. Since each sinusoid is a periodic component of the original signal, the square length of its coefficient defines the power carried by that particular frequency, and signifies the dominance of the periodic component in the original signal. The dominant periodic component can be discovered by examining this power spectrum, also known as the *periodogram* analysis. A periodogram is computed as $P(k) = ||X(k)||^2$, where $k = 0, 1, ... \lceil \frac{N-1}{2} \rceil$. Note that according to the Nyquist sampling theorem, spectral analysis can reliably detect frequencies that are at most half of the maximum signal frequency so that k is at most $\lceil \frac{N-1}{2} \rceil$. Thus, $P(k)$ gives the power at a specific frequency index f_k. After computing the periodogram, one straightforward way to find periodicity is to pick the frequency with maximum power. However, this does not work in practice for the following reasons.

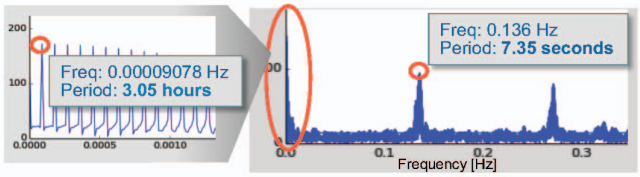

Fig. 4. Periodogram of Conficker botnet traffic

Limitations of Naïve Approach. First of all, DFT can be applied to *any* discrete time series (even those random time series that are not periodic) to decompose it into periodic sinusoids. Consequently, blindly extracting the frequency carrying the largest energy will result in significant false detection.

Second, a time series can consist of multiple periodic behaviors interleaved with each other. For example, Fig. 2 depicting Conficker C&C traffic exhibits two periodic behaviors at two different time granularities. The corresponding periodogram is shown in Fig. 4; selecting only the top frequency in the periodogram may lead to an incomplete picture of the beaconing behavior.

Third, the discrete nature of the DFT yields intrinsic quantization errors and sampling artifacts. As DFT divides the spectrum into N buckets or "bins", each frequency $f_k = k/N$ in the periodogram actually corresponds to a range of periods, between adjacent frequencies. This means that the coefficient for frequency component f_k actually corresponds to periods $[\frac{N}{k}, \frac{N}{k-1})$, meaning that the periodogram has very coarse resolution and deteriorated accuracy for large periods (i.e., small k). Related to this, DFT also suffers from a "spectral leakage" problem, which creates additional periodic components in the frequency domain. Spectral leakage occurs when a signal consists of frequencies that are not exact integer multiples of the DFT frequency bin width. In such cases, the energy of these frequencies will be dispersed or "leaked" from the main frequency bin to the surrounding bins across the spectrum. This results in unwanted side lobes next to the main lobe in the periodogram (e.g., the lower right plot in Fig. 4).

Our Approach. Despite these limitations, a periodiogram still proves useful in automatically extracting a set of candidate frequencies that potentially correspond to the true periodicities. The challenge is to determine an appropriate power threshold such that only the dominant frequencies carry energy higher than the threshold.

To determine this threshold, we apply the permutation-

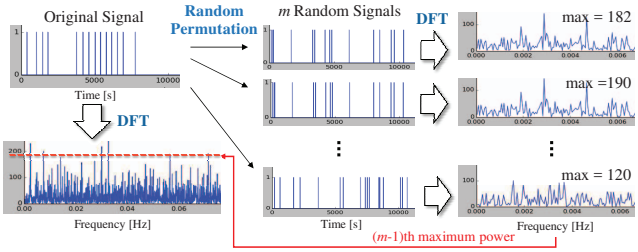

Fig. 5. Permutation based filtering

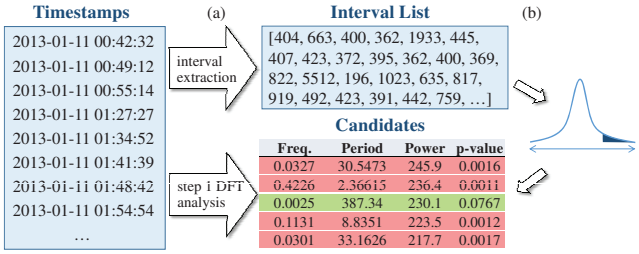

Fig. 6. Pruning using statistical features (TDSS bot)

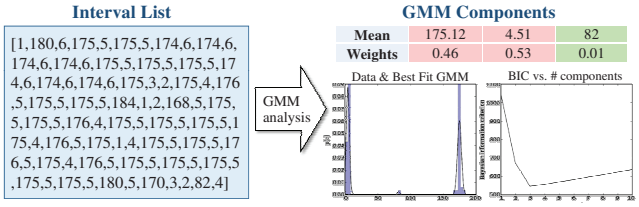

Fig. 7. GMM for detecting multiple periods

based filtering algorithm proposed in [25]. The main idea is to identify how much of the signal energy can be attributed to non-periodic or random mechanisms; then the only frequencies of interest would be those with power higher than ones derived from random processes. Given a time series sequence $x(n)$, the algorithm creates the random permutation $x'(n)$ of all N elements of $x(n)$. This shuffling process would destroy any periodic patterns in $x(n)$ while the first order statistics and characteristics not related to timing persist (e.g., amplitude). Therefore any structural information exhibited in $x'(n)$ is likely from random noises and should be discarded.

Based on this observation, we perform the same peridogram analysis on the permuted signal $x'(n)$, record the maximum power in its frequency domain (denoted as $p_{max}(x')$) and use it as the threshold to identify potential candidate frequencies in the original $x(n)$. In other words, if a frequency in $x(n)$'s periodogram carries less power than the threshold, it is likely due to random noises and thus is eliminated from the candidate frequency set.

One permutation may not completely destroy all periodic structures. To provide a reasonable confidence level C on the power threshold, we repeat the above permutation process m times, and use $(C \times m)$-th highest power frequency (e.g., 19th when $C = 95\%$ and $m = 20$) of $p_{max}(x')$ as a good estimator for the power threshold p_T. The process is illustrated in Fig. 5.

The result of this process is a set of most promising frequencies (denoted as $F = \{\mathcal{F}_1, \mathcal{F}_2, ...\}$) that potentially correspond to true periodicities of $x(n)$. If the set is empty, i.e., none of the frequency exceed the threshold, the original time series $x(n)$ will be rejected and considered as non-periodic.

C. Step 2: Pruning

After obtaining a set of candidate frequencies F and their corresponding periods (denoted as $P = \{\mathcal{P}_1, \mathcal{P}_2, ...\}$) in the time domain, the next step is to further filter out less feasible candidates, and reduce the number of inputs to the more expensive verification step.

High-Frequency Noise. During our analysis, we observed that candidate sets often include high-frequency noise components. To identify such cases, we convert the timestamps T into an *interval list*, i.e., $I = \{i_1, i_2, ..., i_{n-1}\}$, where $i_k = t_{k+1} - t_k$ is the time interval between two consecutive connections (Fig. 6 (a)). Analyzing the statistics of the interval list often allows us to further prune impossible or unlikely candidates. For example, any candidate period $\mathcal{P}_k \in P$ that is smaller than the minimum interval, i.e., $\mathcal{P}_k < min(I)$, is due to high frequency noise and can be removed from the candidate set. Figure 6 shows an example of the TDSS botnet traffic and its interval list where the minimum interval is 196 seconds. Therefore among all the candidate periods from peridogram analysis in step 1, only 387.34 is larger than the minimum interval and all others can be safely removed from the candidate set.

Hypothesis Testing. Extending the threshold-based approach, we design a hypothesis testing algorithm to determine the statistical significance of the candidate periods given the observed intervals. Given candidate period $\mathcal{P} \in P$, we construct our null hypothesis H_0: $\mathcal{P}$ is the true period of the original signal. Due to various noises in the real world scenario (e.g., network delays, retransmission, context switching, etc.), the observed intervals may not be exactly $\mathcal{P}$. Instead, we can model the distribution of the observed intervals as a random variable $N(\mathcal{P}, \sigma^2)$, i.e., the intervals are generated from an underlying normal distribution with mean/variance $(\mathcal{P}, \sigma^2)$.

The goal of hypothesis testing is to decide, given a predefined significance level α and the set of observations, whether to reject the null hypothesis H_0 in favor of the alternative hypothesis H_1: $\mathcal{P}$ is *not* the true period. In our case, we applied a one-sample t-test on the observed intervals and calculated the p-value. We reject $\mathcal{P}$ and remove it from the candidate set if the p-value is smaller than the chosen significance level $\alpha = 5\%$. Take the TDSS botnet case in Figure 6 as an example. All candidate periods except 387 seconds were pruned since their p-values are all considerably smaller than 5%.

One of the major benefits of applying hypothesis testing in the pruning step is its conservativeness, i.e., the null hypothesis is rejected only when there is significant evidence against it.

Sampling Rate. Another pruning criterion is the sampling rate. We filter out time series that are under-sampled. This is particularly important in the rescaling and merging phase (Sect. VII) where fine-grained time series are aggregated into coarser granularity for better scalability and periodicity

detection over longer time period. According to the Nyquist Theorem, one can accurately measure the frequency f of a signal only when the sampling rate is as least $2f$ (called Nyquist frequency). Otherwise, false images of the signal may be created at frequency below f. For example, consider the beaconing behavior at every 30 seconds. If the sampling rate is higher than once every 15 seconds (or data aggregation in buckets of 15 seconds or less), the resulting time series will consist of both peaks ("1") and valleys ("0") that allows accurate measurement of the true frequency. In contrast, when the sampling rate is smaller than the Nyquist frequency (e.g., data aggregation in 20-second buckets), the resulting time series may consist of only peaks, leading to incorrect detection of periodicity. One way to identify a potentially under-sampled signal is to examining the interval lists I and seeing whether most of the intervals are equal to 1. If so, that means the signal has peaks in most time units, indicating the lack of visibility into the signal's inherent structure due to coarse grained sampling (or large aggregation window). In BAYWATCH, we prune the time series if more than 80% of its intervals are equal to 1.

Multiple Periods. Another challenge in mining periodic behavior is that the time series may have multiple periods. For example, Figure 7 shows the interval lists of the time series with two interleaved real periodic behaviors. Because the intervals are separated into two sets (5 seconds and 175 seconds), statistic testing that assumes a *single* underlying model will likely reject every candidate period because the observed distribution cannot be explained by a *single* distribution. In BAYWATCH, we address this challenge by using the Gaussian mixture model (GMM). GMM is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. It uses the iterative expectation maximization (EM) algorithm to estimate the parameters of each Gaussian component. The number of underlying Gaussian components is selected to be the one that minimizes the Bayesian Information Criterion (BIC). Figure 7 shows the histogram of the observed intervals along with the best-fit GMM model. The rightmost plot shows the BIC value as a function of the number of components where its minimum is at three components. The mean (μ) and weight (i.e., percentage of data points) of each Gaussian component are summarized in the table above the plots. The GMM model successfully identifies the two main components[2] indicating that there are likely two interleaved periodicities in the original time series. For such cases, we group intervals according to the their Gaussian component and separately apply statistical testing to each group. We will retain the candidate period if it is not rejected by any interval groups.

D. Step 3: Verification

After the discovery and pruning steps, the remaining candidate set contains a small number of potential periods, likely corresponding to the true periodicity of the original time series. The final step is to verify these candidates' validity with the circular autocorrelation function (ACF) in the time domain. ACF is a useful tool to analyze repeating patterns by calculating the similarity between a time series and itself shifted by a certain time τ. ACF is calculated as

[2]The 3rd component is considered as noise due to its low weight.

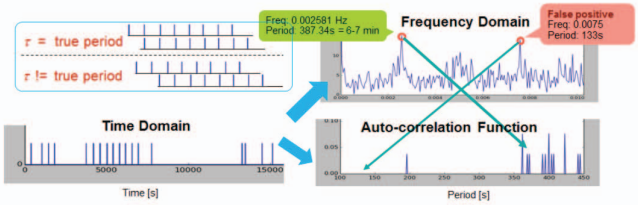

Fig. 8. Verifying candidate periods with ACF

$ACF(\tau) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) \cdot x(n+\tau)$. The intuition is that if τ is the true period, the shifting would create a signal that is off by one period and therefore should largely overlap with the original signal and would reinforce it. Otherwise, the shifted signal interleaves with the original signal, rather than overlapping with it, resulting in a low correlation value.

Essentially ACF provides a more fine-grained periodicity detector. However, ACF alone may be inefficient to detect periodicity because without candidate discovery and pruning steps, the number of possible periods that need to be verified could become very large (i.e., N). Therefore in BAYWATCH, ACF is used as a verification tool and applied only on a small set of most feasible candidate periods (normally less than 5), thus offering much better scalability when analyzing large data sets. Figure 8 illustrates an example of candidate verification using ACF.

V. SUSPICIOUS INDICATOR ANALYSIS

In the third phase, domain knowledge and historical information are taken into account in order to distinguish legitimate beaconing behavior from suspicious behavior. The objective is to reduce the number of cases for the manual investigation and verification (the fourth phase) as much as possible by only reporting cases that have a high confidence of being related to malicious activities.

A. URL Path Token Filter

At this step, we look into the *resource path* within a URL accessed. A resource path often represents the type of the web resource, e.g., "rss" for RSS feeds and "images" for image files, or the structure of the web request, e.g., "search?q=N" for Conficker worms. This analysis is complementary to the whitelisting mechanism discussed in Section III in that whitelisting focuses on a domain name or an IP address in a URL.

In order to filter out likely benign beaconing cases, BAYWATCH measures the popularity of tokens in a resource path, where tokens are extracted by splitting a resource path using the delimiter "/". Similar to whitelists, a web request containing known and popular tokens is not likely a candidate for malicious beaconing.

BAYWATCH identifies popular tokens from the request history of the previous n days $(D-n, ..., D-1)$ to filter out likely benign cases on day D. This moving window approach yields a dynamic list of popular tokens reflecting the latest trends, which significantly reduces the chances that an attacker may abuse a static list of well-known tokens to avoid detection.

B. Novelty Analysis

The goal of this filter is to eliminate duplicate work by the analyst. More specifically, it removes source/destination pair candidates that have been reported for beaconing behavior already, or at least whose destination has been involved in pairs already reported, either from another candidate or from a previous analysis run. The candidate is still logged and reported, and is kept available for analyst review; however, it will not be passed into the ranking algorithm for later manual verification and investigation.

This filter can be considered as "change detection," consolidating cases of the same source/destination pairs and thereby forwarding cases only when a destination has not been reported before, or a source has not been reported before as beaconing to that destination.

C. Language Model Score

A common strategy employed by botnets and malicious servers to circumvent detection is domain name generation. Bots algorithmically create a large pool of random names and attempt to connect to them, hoping that some domains may have been registered by attackers in advance as a rendezvous point. To avoid collision with existing domains, randomly generated names typically exhibit distinct patterns, e.g., combination of characters that rarely occur in popular domain names. Such patterns often provide good indication of suspicious nature.

Motivated by this observation, we train a 3-gram language model using Alexa top 1 million domain names [1]. An n-gram model is a probabilistic model that can be used to predict the next character in the sequence given previous $n-1$ characters. More specifically, given a training corpus of domain names, the algorithm first extracts all the n-gram substrings $c_1c_2\ldots c_n$, and compute the transitional probability, $P(c_n|c_1c_2\ldots c_{n-1}) = \frac{\text{Count}(c_1c_2\ldots c_n)}{\text{Count}(c_1c_2\ldots c_{n-1})}$. This is the probability of c_n appearing after $c_1c_2\ldots c_{n-1}$. With the derived transitional probability from the training corpus, we compute the score S of domain name $D = c_1c_2\ldots c_m$ under the n-gram language model as $S = log(P(D))$, where $P(D) = \Pi_{k=n}^{m} P(c_k|c_{k-n+1}c_{k-n+2}\ldots c_{k-1})$[3]. A low score means abnormal or random patterns in a domain name, which may warrant further investigation. For example, botnets using domain generation algorithms (DGAs) often yield low scores. The score of domain skmnikrzhrrzcjcxwfprgt.com is -45.166, which is significantly lower than -7.406378 of google.com.

D. Weighted Result Ranking

To assist analysts in prioritizing their investigation efforts, we combine various indicators, e.g., periodicity strength, language model score, and destination popularity, into a single weighted ranking score. The weight of each indicator can be empirically set based on the quality of the indicator and preference of the analysts. For example, in the current implementation of BAYWATCH, we assign a higher weight to the language model score for the domains with very low probabilities. We also awarded a higher score to the connections with strong periodicity, e.g., high ACF score, low standard deviation in the observed intervals, and periodic over long range of time, since these regular patterns are of more interest to the analysts. For the final ranking of suspicious beaconing cases, we apply a threshold to report only those cases that exceed the n-th percentile of the score distribution.

[3]Kneser-Ney smoothing is used for previously unseen n-grams.

TABLE II. LIST OF FEATURES USED IN TRAINING CLASSIFIERS

Feature	Definition
series length	# intervals in series
period(s)	most dominant period(s)
power	power of most dominant period(s)
similar source	# sources sharing same destination
n-gram count	hist. of n-grams in symbolized series
entropy	entropy of symbolized series
compressibility	compression ratio of symbolized series

VI. INVESTIGATION AND VERIFICATION

In the aforementioned phases, we apply various filtering mechanisms to triage detected cases. For large networks, it is highly likely that a large number of suspicious cases pass through the triage for further manual investigation. Instead of an exhaustive investigation, we propose the following alternative: (a) sampling a small set of candidate cases and manually investigating them, (b) using these cases and their corresponding diagnosis results as "ground truth" to train a classifier, and (c) applying the trained classifier to classify the remaining candidate cases. For example, in our evaluation (see Sect. VIII), we manually investigate a data set collected over a one-month time interval to train a classifier, and apply it to a much larger data set spanning over a five-month time period.

Astute readers might ask why this approach is not applied in earlier phases. The reasons are twofold. First, at this stage, the candidate cases are much "cleaner" for training a classifier because many noisy cases have been pruned. Second, as a desired side effect, the various filtering mechanisms essentially generate a rich set of features (e.g., dominant periods in Sect. IV). Next we elaborate the feature set and the classifier we use in our prototype.

A. Feature Set

Recall that each candidate case is a tuple $\langle s, d, (i_1, i_2, \ldots)\rangle$ with s, d, and $(i_1, i_2, \ldots)$ representing the source, destination, and a series of time intervals, respectively. For each case, we generate the set of features as defined in Table II. Among them, the first four features are fairly straightforward. We concentrate our discussion on the remaining three.

Once we detect the most promising period(s) as in Section IV, we "symbolize" the series of time intervals by applying the following rule for each interval i:

$$\begin{cases} i \rightarrow \text{'x'} & i \text{ appears in dominant period(s)} \\ i \rightarrow \text{'y'} & i = 0 \\ i \rightarrow \text{'z'} & \text{otherwise} \end{cases}$$

Since this simplified sequence comprises only three letters, we can easily measure its entropy, its n-gram histogram ($n = 3$), and its compressibility. In particular, the compressibility of a symbolized series is measured by its compression ratio by "gzip" under the highest compression level.

B. Classifier

In our prototype, we adopt random forest [5] as a classifier. A random forest is a collection (or ensemble) of decision trees. When classifying a new case, each decision tree assigns the case to a single class (e.g., benign or malicious); the output of the random forest is the mode of the outputs of the decision trees. The random forest classifier has a number of desired features for our context: (a) built upon decision trees, it can handle both numeric and categorical features; (b) it has been empirically shown to outperform, in terms of accuracy, both SVM and naïve Bayes classifiers across a wide range of data sets; and (c) it is highly scalable and inherently parallelizable.

VII. Implementation

We implemented BAYWATCH in 4,200 lines of Java on top of the MapReduce framework [7] to efficiently process large-scale data. Each phase is designed in a modularized MapReduce job to avoid reprocessing raw logs.

A. Data Extraction

From raw logs stored in HDFS, e.g., web proxy logs, and DNS logs, BAYWATCH first extracts request time intervals per communication pair. By default, time intervals are extracted at the finest granularity, e.g., a second level.

- MAP: $\langle k, l\rangle \rightarrow \langle H(s,d), (s,d,ts)\rangle$. For each input line l, a MAP task identifies source s, destination d, and timestamp ts to extract request information. A hash function H is used to control the number of REDUCE tasks and output files. For example, a 5-bit hash results in 32 ($= 2^5$) REDUCE tasks and output files, helping to minimize the startup overhead arising from too many REDUCE tasks and too much disk I/O. MapReduce performs better with a smaller number of larger files rather than a larger number of smaller files.
- REDUCE: $\langle H(s,d), (s,d,ts)\rangle$ list $\rightarrow \langle k, AS\rangle$. As the hash of s and d is the key, request timestamps of the same source/destination pair are grouped and sorted to get the list of request timestamps. From the list, a REDUCE task obtains request intervals per communication pair. The output value of REDUCE is ActivitySummary which summarizes request activities of every source/destination pair. ActivitySummary AS consists of the following: source/destination pair $s:d$, time scale e (1 second at the finest granularity), the first request timestamp ts_1, and the list of request intervals int along with side-channel information (such as full URLs) for the token filter in Section V-A.

B. Rescaling and Merging

This phase provides BAYWATCH with scalability and flexibility. Instead of reprocessing raw logs, BAYWATCH rescales ActivitySummary from the previous phase to improve periodicity detection performance over long time ranges, and aggregates request history to reduce data size.

- MAP: $\langle k, AS\rangle \rightarrow \langle H(s,d), AS_{rescaled}\rangle$. Given new time scale e', a MAP task rescales old intervals int to new intervals int', and finds new first time stamp ts'_1 corresponding to the new time scale. Rescaled ActivitySummary $AS_{rescaled}$ is constructed. Hash function H is used to reduce REDUCE and disk I/O overhead.
- REDUCE: $\langle H(s,d), AS_{rescaled}\rangle$ list $\rightarrow \langle k, AS_{merged}\rangle$. A REDUCE task merges multiple intervals from $AS_{rescaled}$ into one per source/destination pair, and yields rescaled and merged ActivitySummary AS_{merged}.

C. Destination Popularity Statistics

As discussed in Section III-B, BAYWATCH leverages local whitelists tuned for a specific organization, which is measured by the number of distinct source features.

- MAP: $\langle k, AS\rangle \rightarrow \langle d, s\rangle$. A MAP task uses the source/destination pair from the ActivitySummary AS, and yields outputs by setting destination d as key and source s as value aggregating sources accessing the same destination.
- REDUCE: $\langle d, s\rangle$ list $\rightarrow \langle d, \mathbb{R}\rangle$. A REDUCE task measures popularity $r \in \mathbb{R}$ of each destination d by calculating the number of sources making a request to a destination, divided by the total number of sources.

D. Beaconing Detection

The beaconing detection phase employs the algorithm described in Section IV to detect periodic requests in an ActivitySummary produced by the data extraction phase (Section VII-A) or the rescaling/merging phase (Section VII-B). If required, request histories are aggregated per source/destination pair.

- MAP: $\langle k, AS\rangle \rightarrow \langle H(s,d), AS_{filtered}\rangle$. A MAP task separates communication pairs.
- REDUCE: $\langle H(s,d), AS_{filtered}\rangle \rightarrow \langle k, (AS, CP)\rangle$. A REDUCE task performs the algorithm based on the request history of every source/destination pair. When periodic behaviors are identified, BAYWATCH reports such ActivitySummary AS with a list of CandidatePeriods CP for subsequent ranking and investigation phases. CandidatePeriod CP consists of the followings: frequency, period, power, auto-correlation score, and domain name score.

E. Ranking

In order to help analysts to prioritize deep investigation, detected periodic behaviors are ranked based on its periodicity and potential maliciousness.

- MAP: $\langle k, (AS, CP)\rangle \rightarrow \langle \mathbb{R}, (AS, CP)\rangle$. A MAP task filters out likely benign beaconing cases based on URL path token analysis (Section V-A) and novelty analysis (Section V-B). Then, periodicity score $p \in \mathbb{R}$ is calculated to rank and weight beaconing cases based on a language model trained on Alexa top 1 million domains, popularity score, and periodicity strength.
- REDUCE: $\langle \mathbb{R}, (AS, CP)\rangle$ list $\rightarrow$ sorted $\langle \mathbb{R}, (AS, CP)\rangle$ list. A REDUCE task collects periodicity scores, and sorts detected beaconing cases to generate a ranked list.

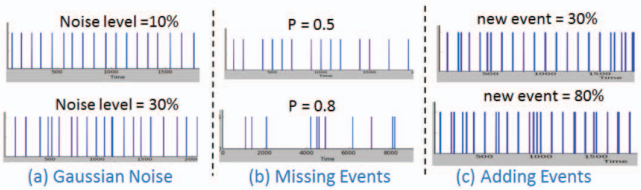

Fig. 9. Different types of noises

VIII. Evaluation

In this section, we provide the results of a systematic evaluation of the periodicity detection algorithm and an evaluation of BAYWATCH on real-world data sets.

A. Noise Tolerant Capability of Algorithm

In this section, we evaluate the performance of the beaconing detection algorithm against different types of noise. The baseline is a periodic time series with frequency $f = 0.01$Hz (period $\mathcal{P} = 100$ seconds). We denote the timestamp of each event as $T(i)$, where $i = 1, ..., N$ and N is the total number of events; we remark that $T(i) - T(i-1) = \mathcal{P}$. We apply three types of noise into the time series (cf. Fig. 9).

The first noise type is the *Gaussian noise*, where for each $T(i)$, we apply a noise value drawn from a normal distribution $N(0, \sigma^2)$. We apply a noise value by shifting the request in time. The standard deviation σ controls the magnitude of the injected noise and we vary it from 0% (no noise) to 100% of period $\mathcal{P}$. This models the noise introduced by factors, such as network delay or the adversary to avoid strictly periodic beaconing. Figure 9(a) shows the time series with 10% and 30% noise level. The second noise type is the *missing-event noise* where each event is associated with a probability p of being dropped. In other words, on average $N * p$ events will be removed from the time series to simulate the noise caused by network failure, system shutdown, etc. The third noise type is the *add-on noise*, where we inject new events at random locations of the time series. The number of injected events quantifies the noise level and is set to be certain percent of the total number of events in the time series, i.e., N. Figure 9(c) shows the time series with 30% and 80% more events injected.

Given a specific noise type and noise level, we ran the beaconing detection algorithm on the noisy time series generated according to the noise model, and reported its periodicity $\mathcal{P}_{noise}$. We repeated the experiments 100 times (each time a new time series is generated with the same noise type and level) and measured the algorithm's performance using the following two metrics: detection ratio γ_d and average difference δ_d between the detected and the true period. Assuming out of the 100 repetitions, the detection algorithm successfully identifies periodicity m times, then γ_d and δ_d can be computed as: $\gamma_d = \frac{m}{100}$, $\delta_d = \frac{1}{m}\sum_{i=1}^{m} |\mathcal{P}_{noise}(i) - \mathcal{P}|$. Essentially, γ_d measures how likely the beaconing algorithm can still detect periodicity under various noise, and δ_d measures the average discrepancy between the true and the detected periods.

Figure 10 summarizes the performance of our beaconing detection algorithm under each of the 3 noise types, as well as under multiple noises together. From Figure 10(a) shows that when the noise level was below 15% of $\mathcal{P}$), the algorithm

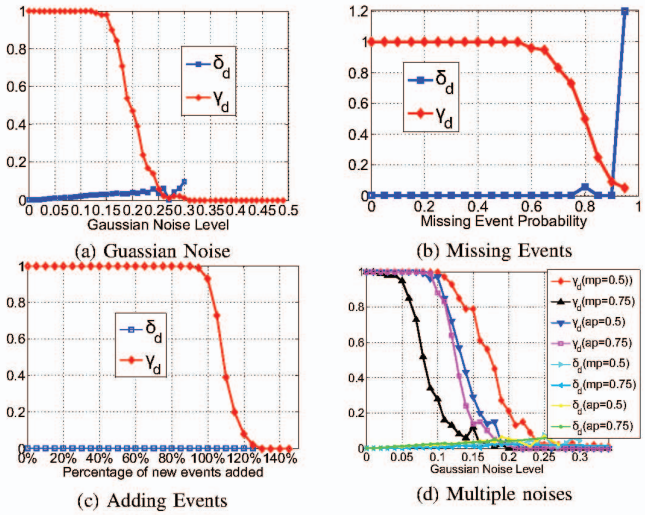

Fig. 10. Performance against different types of noise. mp: missing-event probability; ap: adding-event percent

was able to detect the periodicity with 100% detection rate and very high accuracy ($\delta_d < 2\%$). As the noise level increased, the detection rate gradually decreased due to the distortion introduced by stronger noise. However, a salient property of our algorithm is that even with the decreased detection rate, the algorithm still detected true period with high precision ($\delta_d < 5\%$) when it identified periodic events. As the noise level increased above 30% of $\mathcal{P}$, where the injected noise likely destroyed the inherent periodic patterns (see Figure 9(a)), the algorithm did not report any periodicity, or only reported periodicity with very small probability (1 or 2 out of 100 repetitions).

Figure 10(b) shows that the algorithm performs extremely well against missing events. It detected the true period with 100% detection rate when the probability of dropping event is smaller than 0.6. The detection rate gradually decreased as more events are dropped from the time series. However, whenever the algorithm was able to identify the periodicity, it identified very close to the true period ($\delta_d < 1\%$). The rationale behind the algorithm's high tolerance against missing events is that the events were dropped randomly (as opposed to uniformly) and there would likely remain a few clusters of periodic events (see Figure 9(b)). When significant amounts of events were dropped (e.g., 90%), the algorithm did not detect the periodicity or occasionally detected a large period value due to the enlarged gaps caused by missing events.

A similar trend can be observed from Figure 10(c), which shows the robustness of the algorithm against extra events. Since adding new events typically does not disturb the original periodic signal, the original periodic component still likely exists in the frequency domain. The newly added events essentially created noise in the high frequency spectrum, which was effectively filtered by the pruning step of the algorithm. When the noise increased and its associated periodic components started to carry enough power (e.g., when the number of injected new events dominated the number of original events), the algorithm rejected the noisy time series as non-periodic.

Finally, we measured the algorithm's tolerance against

TABLE III. DATA VOLUMES OF WEB PROXY LOGS

Month	Log Size (gzipped)	# Events
Oct 2013	1.2 TB (0.4 TB)	0.6 B
Nov 2014	6.4 TB (0.9 TB)	6.4 B
Dec 2014	6.4 TB (0.9 TB)	6.3 B
Jan 2015	7.4 TB (1.1 TB)	7.5 B
Feb 2015	6.5 TB (1.0 TB)	6.3 B
Mar 2015	7.7 TB (1.1 TB)	7.4 B
Total	35.6 TB (5.3 TB)	34.6 B

multiple noises injected into the baseline time series. Specifically, we first applied the missing-event or adding-event model to bring noise into the time series. Then we measured δ_d and γ_d against varying levels of Gaussian noise. The results illustrated in Figure 10(d) generally follow similar pattern as Figure 10(a). A noticeable difference is that the threshold, where the algorithm reliably identified periodicities in the noisy time series, dropped from 30 in Figure 10(a) to around 11 and 7 in Figure 10(d). The algorithm performed the worst with the combination of the Gaussian noise and the missing-event noise with probability 0.75.

Nevertheless, the evaluation shows that even with multiple types of noise, the detection algorithm still achieved high accuracy ($\delta_d < 5\%$) when the noise level was not strong enough to significantly distort the periodic pattern.

B. Beaconing in Web Proxy Logs

In order to evaluate the applicability of BAYWATCH in real-world data sets, we performed beaconing detection analytics on web proxy logs collected at a large corporate network. Web proxy logs are good reference data sets to evaluate the performance and deployment of BAYWATCH since web traffic is often misused by adversaries to disguise their malicious payload in a huge amount of legitimate traffic, which crosses the perimeter. Although organizations typically employ web traffic monitoring methods, it is not trivial to analyze all events historically to identify beaconing behaviors due to the size and combinatorial complexity of the collected data.

1) Environment & Data sets: The web proxy logs from a BlueCoat ProxySG infrastructure have been collected at two time intervals: a consecutive 5-month time period from November 2014 to March 2015, and a 10-day time period in October 2013. Table III provides a detailed breakdown of the data volumes. The proxy logs were positioned at the perimeter of a multi-site corporate network; for each log entry, we correlated the source IP address with the MAC address obtained via the centralized DHCP server log repository. Compared to an IP address, a MAC address is more reliable in device identification because IPs may change over time as a device can connect from different networks. Overall, we observed over 240 K IP addresses and 130 K distinct MAC addresses.

We evaluated BAYWATCH on a MapReduce framework consisting of 13 nodes, each equipped with 16–24 cores, 96 GB memory, and 1 TB disk. Each node is also configured to run up to 6 map tasks and up to 4 reduce tasks at once. In our beaconing detection experiments, we defined a connection pair such that a source MAC/IP was the source feature, and a destination domain/IP was the destination feature.

TABLE IV. CONFUSION MATRIX OF CASE CLASSIFICATION

	classified benign	classified malicious
true benign	2163	0
true malicious	41	148

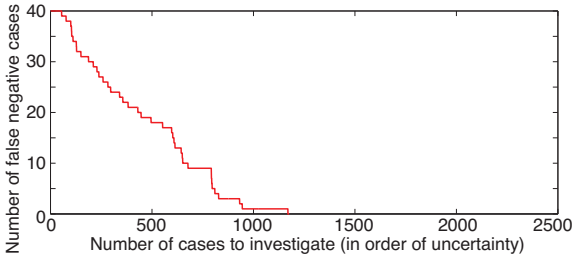

Fig. 11. Number of false negative cases

2) Results: Daily analysis (5-month trace): We have run the complete data-flow depicted in Fig. 3 over the 5-month trace while choosing a resolution of 1 second for the event time stamps when building the activity summaries (cf. Section VII-A). The time series analysis has been run over daily intervals to simulate daily operations of BAYWATCH. We chose a local whitelist threshold $\tau_P = 0.01$ (1% of the population), and the threshold over the score distribution in the ranking filter has been chosen at the 90th percentile.

BAYWATCH took about 35 hours to process one month of proxy logs, e.g., March 2015 data set: extracting ActivitySummary of every connection pair from raw logs of each day, detecting periodic events, and generating a ranked list of CandidatePeriod. Runtime mainly depended on the amount of data to be analyzed, especially the number of connection pairs. During weekends, on average, there were 3.3 million distinct connection pairs, and it took 14 minutes to complete the entire analytics steps. During weekdays, on average, there were 26 million distinct connection pairs, and it took 1 hour and 30 minutes to complete the analytics. It demonstrated that our implementation was scalable enough to analyze millions of connections per day to detect surreptitious beaconing activities.

In total, 2,352 distinct destinations were flagged as suspicious. Manually examining all these cases would have been extremely expensive. We thus applied the bootstrap approach proposed in Section VI. More specifically, we examined a much smaller data set collected over January 2015 (see Table III). From these cases, we derived the features defined in Table II, and used these feature vectors as well as their labels (0 for 'benign' and 1 for 'malicious') to train a random forest classifier (consisting of 200 decision trees). We then applied this classifier to the rest of the cases of the 5-month trace.

To evaluate the effectiveness of this approach, we queried VirusTotal with all of the involved destinations and used the returned reports to construct the "ground truth": specifically, if any of its anti-virus engine reported the destination as malicious, we labeled it as malicious. The confusion matrix in Table IV shows the classification results. Among the 2,352 cases, a majority of them were correctly classified. In particular, the classifier was able to achieve a false positive rate of 0 with respect to VirusTotal labels. As we examined the 41 false negative cases in Table IV, we found indeed many of them were of low certainty by the classifier. One possible

TABLE V. EXAMPLE CASES FOUND IN 5-MONTH TRACE

Domain name	Smallest period	Clients
cdn.5f75b1c54f8[..]2d4.com	30 seconds	19
img.ddbd60eeb01[..]cce.com	901 seconds	1
b117f8da23446a1[..]92a.pl	929 seconds	3
www.iiasdomk1m9[..]4z3.com	165 seconds	2

solution is therefore: (a) ranking candidate cases based on their uncertainty, and (b) manually examining the set of most uncertain cases. Figure 11 shows that the number of false negative cases is reduced quickly as we examine candidate cases according to their uncertainty order. For example, after checking around 550 cases, the number of false negative cases was already below 10.

We manually investigated the 50 top-ranked destinations classified as malicious. This included a comprehensive search for any indications on security intelligence platforms or threat reports, e.g., public portals such as VirusTotal, McAfee Site Advisor, CleanMX, or IBM X-Force Exchange. For 48 out of the 50 (96%) top-ranked destinations, we found clear evidence of using the domains for malicious activity. Most of them were reported for distributing malware or adware (e.g., Trojan-Downloader.Win32.Genome, Trojan.Semnager, etc.) or Android APK files exploiting vulnerabilities. For some of the destinations, 20 or more unique source identifiers (i.e., MAC addresses) were observed—amounting to 93 distinct clients in the confirmed top 50. Beaconing periods detected in the top 50 varied greatly and ranged between 30 seconds and 929 seconds (cf. Table V).

While further investigating the top-100 destinations reported as malicious, we found a few false positives: One destination (api.echoenabled.com) had dozens of clients; according to our findings this destination relates to "HTTPS Everywhere," a browser plugin from the Electronic Frontier Foundation (EFF). Some other destinations were related to ad networks. In addition, we have found some truly legitimate cases such as sports, news, and music streaming websites (e.g., 2015.ausopen.com, kdfc.web-playlist.org), which presumably updated contents on a regular basis and were used by users over longer time periods—and were not filtered by our initial whitelisting mechanisms.

TABLE VI. TOP 5 CASES REPORTED IN 10-DAY TRACE

Rank	Domain name	Smallest period	Clients
1	setup.poiio[..]rew.com	180 seconds	1
2	setup.ghwr8[..]vsh.com	180 seconds	1
3	cuoxxscrhhv[..]igp.com	63 seconds	3
4	akkixiodzmf[..]asp.com	63 seconds	1
5	vzsjfnjwchf[..]hxa.com	1242 seconds	1

3) Results: Daily analysis (10-day trace): The same parameters as in Section VIII-B2 were applied to the 10-day trace collected in October 2013. In this trace, we were aware of several confirmed positives (related to botnet infection due to the ZeuS.Zbot malware [22]). For a total of 828 communication pairs the traffic conditions indicated suspicious beaconing behavior: they can be broken down into 412 unique destinations and 696 unique clients. Among the cases, we manually investigated the 5 top-ranked destinations. Using public threat intelligence portals, we were able to confirm that the domains (cf. Table VI) were related to malware activity: beaconing of Zeus.Zbot (top 1 and 2), and beaconing of ZeroAccess (top 3, 4, and 5).

IX. RELATED WORK

Botnets and recent advanced threats are often orchestrated through C&C infrastructures. Many approaches have been proposed to detect botnets through network traffic analysis. For example, BotHunter [11] identified a typical bot infection lifecycle by monitoring network traffic, such as scanning, exploit download. BotSniffer [12] leveraged the temporal and spatial correlation, and similarity between multiple infected hosts to detect botnet activities. JackStraws [14] took advantage of host-based information to augment network behavior graphs to provide more complete picture, and captured potential C&C traffic. A similar idea was leveraged in [26] for botnet detection. Similar to our work, Giroire et al. [10] proposed to use temporal persistence to detect covert botnet traffic. Tegeler et al. [24] developed BotFinder which applied machine learning algorithm on various temporal features (e.g., interval list, FFT), and built models on known botnet samples. AsSadhan and Moura [3] analyzed control plane traffic looking for periodic behavior that might indicate the presence of botnet. Despite the similarity of using temporal features to detect suspicious network activities, the major difference is that our work does not require known training data set or the models of botnet behavior. Moreover, our focus is on reducing false positives and ensuring the practicality of the system, e.g., daily operation in a large enterprise network. Many of our design choices (e.g., filtering, ranking, and investigation) are derived by analysts' feedbacks.

Periodicity detection has also become a popular topic in the data mining and database research communities because it can be used to capture the evolution of data over time. Earlier work focused on mining frequent patterns in the time series such as [13]. The goal of the work was to find patterns that appeared at least min_sup times. Later research addressed the problem of automatic periodicity detection. For example, Elfeky et al. [8] proposed an efficient convolution method to detect periodicity. More recently, Rasheed et al. [19] proposed a periodicity mining method in time series databases using suffix tree. The algorithm was noise-resilient and ran in quadratic time in the worst case. Elfeky et al. [9] used time warping techniques to accommodate insertion or delete noises in the time series data, and achieved $O(n^2)$ complexity. Vlachos et al. [25] combined two complementary approaches, such as frequency analysis and autocorrelation, and designed a non-parametric method for accurate periodicity detection. Due to its lower computational cost, i.e., $O(n \log n)$, BAYWATCH adapted Vlachos' approach and scaled it to be practical for a massive amount of network traffic.

X. DISCUSSION

In addition to web proxy logs, BAYWATCH is applicable to other data sources such as DNS, Netflow, or firewall logs. The core of the methodology, the time series analysis, relies on the activity summary of a communication pair, including a source, a destination, and request time intervals. In our preliminary experiments on Netflow and DNS, we identified numerous suspicious beaconing behaviors; however, we omit the results here due to the space limit. In DNS, challenges with detecting

beaconing are that regional DNS servers may see aggregated behaviors of local DNS servers rather than end hosts, and may not see every DNS query due to caching. Netflow only provides connection level information, i.e., no domain names or additional content information.

We operate BAYWATCH iteratively in intervals at three time scales (daily, weekly, monthly). This allows BAYWATCH to detect less-frequent beaconing (e.g., 24-hour periodicity) as well as highly-frequent beaconing (e.g., minute-level periodicity). Even though BAYWATCH is designed to be greatly robust against nosies in data sources, adversaries may still be able to evade detection by employing purely random behavior. However, this may be impractical since it imposes substantial maintenance cost on adversaries. For example, it is unpredictable for adversaries when infected hosts will call back (soldier without discipline), and there is no guarantee when their commands or updates will be conveyed to the botnets.

We systematically evaluated the noise tolerance of our beaconing detection algorithm with synthetic data sets by injecting various noises. However, it is not typically straightforward to measure the detection accuracy (distinction between legitimate and malicious beaconing) at a large scale on real-world data due to the lack of ground truth. Since security analysts face the same challenge, our filtering approach to reduce the workload, and to output a prioritization list proved to be beneficial.

XI. CONCLUSIONS

In this paper we proposed a robust and scalable methodology, BAYWATCH, to identify stealthy beaconing behavior from passive network traffic monitoring. Our experimental results with both synthetic data and 35TB of real-world logs demonstrated robustness against real-world perturbations, scalability to millions of connections per day, and accuracy in detecting malicious beaconing.

ACKNOWLEDGMENTS

We would like to thank the anonymous referees and Doug Kimelman for their valuable suggestions and helpful comments.

REFERENCES

[1] Alexa Internet, Inc. Does Alexa have a list of its top-ranked websites? https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-.

[2] Arbor Networks. Twitter-based botnet command channel, August 2009. https://asert.arbornetworks.com/twitter-based-botnet-command-channel/.

[3] B. AsSadhan and J. M. Moura. An efficient method to detect periodic behavior in botnet traffic by analyzing control plane traffic. *Journal of Advanced Research*, 5(4):435 – 448, 2014.

[4] B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi. Duqu: A Stuxnet-like malware found in the wild, October 2011. http://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf.

[5] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001.

[6] K. Burton. The Conficker Worm, 2010. https://www.sans.org/security-resources/malwarefaq/conficker-worm.php.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[8] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid. Periodicity detection in time series databases. *Knowledge and Data Engineering, IEEE Transactions on*, 17(7):875–887, 2005.

[9] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid. Warp: time warping for periodicity detection. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[10] F. Giroire, J. Chandrashekar, N. Taft, E. M. Schooler, and D. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Proceedings of Recent Advances in Intrusion Detection (RAID)*, 2009.

[11] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium*, 2007.

[12] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[13] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 106–115. IEEE, 1999.

[14] G. Jacob, R. Hund, C. Kruegel, and T. Holz. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *Proceedings of the 20th USENIX Conference on Security (SEC '11)*, 2011.

[15] Kaspersky Lab Global Research and Analysis Team. The Epic Turla Operation: Solving some of the mysteries of Snake/Uroboros, August 2014. https://securelist.com/files/2014/08/KL_Epic_Turla_Technical_Appendix_20140806.pdf.

[16] Kaspersky Labs. The Epic Snake: Unraveling the mysteries of the Turla cyber-espionage campaign, August 2014. http://www.kaspersky.com/about/news/virus/2014/Unraveling-mysteries-of-Turla-cyber-espionage-campaign.

[17] B. Krebs. Sources: Target Investigating Data Breach. Krebs on Security, December 2013. http://krebsonsecurity.com/2013/12/sources-target-investigating-data-breach/.

[18] Mandiant. APT1 – Exposing One of China's Cyber Espionage Units, February 2013. http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf.

[19] F. Rasheed, M. Alshalalfa, and R. Alhajj. Efficient periodicity mining in time series databases using suffix trees. *Knowledge and Data Engineering, IEEE Transactions on*, 23(1):79–94, 2011.

[20] A. Rassokhin and D. Oleksyuk. TDSS botnet: full disclosure, March 2012. http://nobunkum.ru/analytics/en-tdss-botnet.

[21] RSA FraudAction Research Labs. Anatomy of an Attack, April 2011. https://blogs.rsa.com/anatomy-of-an-attack/.

[22] Symantec. Trojan.Zbot, January 2010. http://www.symantec.com/security_response/writeup.jsp?docid=2010-011016-3514-99&tabid=2.

[23] Symantec Security Response. Regin: Top-tier espionage tool enables stealthy surveillance, November 2014. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/regin-analysis.pdf.

[24] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2012.

[25] M. Vlachos, S. Y. Philip, and V. Castelli. On periodicity detection and structural periodic similarity. In *Proceedings of SIAM International Conference on Data Mining (SDM)*, 2005.

[26] Y. Zeng, X. Hu, and K. G. Shin. Detection of botnets using combined host- and network-level information. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2010.

[27] K. Zetter. Sony Got Hacked Hard: What We Know and Don't Know So Far. Wired, December 2014. http://www.wired.com/2014/12/sony-hack-what-we-know/.