Detecting Malicious Domains via Graph Inference

Pratyusa K. Manadhata[1], Sandeep Yadav[2], Prasad Rao[1], and William Horne[1]

[1] Hewlett-Packard Laboratories
[2] Damballa Inc.
{pratyusa.k.manadhata,prasad.rao,william.horne}@hp.com,
sandeepvaday@gmail.com

Abstract. Enterprises routinely collect terabytes of security relevant data, e.g., network logs and application logs, for several reasons such as cheaper storage, forensic analysis, and regulatory compliance. Analyzing these big data sets to identify actionable security information and hence to improve enterprise security, however, is a relatively unexplored area. In this paper, we introduce a system to detect malicious domains accessed by an enterprise's hosts from the enterprise's HTTP proxy logs. Specifically, we model the detection problem as a graph inference problem- we construct a host-domain graph from proxy logs, seed the graph with minimal ground truth information, and then use belief propagation to estimate the marginal probability of a domain being malicious. Our experiments on data collected at a global enterprise show that our approach scales well, achieves high detection rates with low false positive rates, and identifies previously unknown malicious domains when compared with state-of-the-art systems. Since malware infections inside an enterprise spread primarily via malware domain accesses, our approach can be used to detect and prevent malware infections.

Keywords: belief propagation, big data analysis for security, graph inference, malicious domain detection.

1 Introduction

This is the age of big data. Organizations collect and analyze large datasets about their operations to find otherwise difficult or impossible to obtain information and insight. Big data analysis has had an impact on online advertising, recommender systems, search engines, and social networks in the last decade, and has the potential to impact education, health care, scientific research, and transportation [1]. *Big data analysis for security*, i.e., the collection, storage, and analysis of large data sets to extract actionable security information, however, is a relatively unexplored area.

Organizations, especially business enterprises, collect and store event logs generated by hardware devices and software applications in their networks. For example, firewalls log information about suspicious network traffic; and hypertext

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part I, LNCS 8712, pp. 1–18, 2014.

transfer protocol (HTTP) proxy servers log websites (or domains) accessed by hosts in an enterprise. Enterprises collect and store event logs primarily for two reasons: need for regulatory compliance and post-hoc forensic analysis to detect security breaches. Also, availability of cheap storage has facilitated large scale log collection. Such logs generated by both security products and non-security infrastructure elements are a treasure trove of security information. For example, if hosts in an enterprise network are infected with bots, then the bots may contact their command and control (C&C) server over domain name system (DNS) and may exfiltrate sensitive data over HTTP. Hence both DNS logs and HTTP proxy logs will contain information about bot activities. Developing scalable and accurate techniques for detecting threats from logs, however, is a difficult problem [2]; we review related work in Section 5. In this paper, we introduce a big data analysis approach to detect malicious domains accessed by hosts in an enterprise from the enterprise's event logs.

1.1 Malicious Domain Detection

Malware infections spread via many vectors such as drive-by downloads, removable drives, and social engineering. Malicious domain access, however, account for majority of host infections [3]. Malware installed on hosts may be involved in pilfering sensitive data, spreading infections, DDoS attacks, and spamming. Legal issues and loss of intellectual property, money, and reputation due to malware activities make security a pressing issue for enterprises. Hence to contain malware, enterprises must prevent their hosts from accessing malicious domains.

Reliable and scalable detection of malicious domains, however, is challenging. Many enterprises use both commercial and freely available domain blacklists, i.e., list of known malicious domains, to detect and prevent malicious domain access; such lists, however, incur a significant delay in adding new domains as they rely on many manual and automated sources. Moreover, the techniques used to generate the lists are resource intensive. For example, malicious domain inference using DNS and network properties such as a domain's IP addresses and BGP prefixes requires data collection from sources with specialized vantage points. Similarly, machine learning techniques, e.g., analyzing a domain's lexical features, or a related IP address's structural properties, requires large feature data sets and accurately labeled training sets; hence these techniques are computationally expensive and may suffer from increased delay in detection.

In this paper, we present a scalable malicious domain detection approach that uses event logs routinely collected by enterprises and requires no additional data collection, and uses minimal training data. We model the detection problem as an *inference* problem on very large graphs. Our graph inference approach utilizes inherent malware communication structure, e.g., all bots in an enterprise contact the same command and control server. We first construct a *host-domain graph* by adding edges between each host in the enterprise and the domains visited by the host. We *seed* the graph with ground truth information about a small fraction of domains obtained from domain blacklists and whitelists, i.e., we label a small fraction of domains as malicious and benign, and label the rest of the domains

as unknown. We then adapt *belief propagation* to estimate an unknown domain's likelihood of being malicious [4,5]; if the likelihood is more than a threshold, we identify the domain to be malicious. We chose belief propagation because it is a fast and approximate estimation algorithm that scales well to large graphs, and also takes structural advantage of malware communication (Please see Section 2 for details).

We applied our approach to 7 months of HTTP proxy data collected at a large global enterprise. Our results show that with minimal ground truth information, e.g., with only 1.45% nodes in the graph, we achieve high true positive rates (TPR) of 95.2%, with low false positive rates (FPR), for 0.68%. A benign node labeled as malicious by our approach is a false positive (FP) and a correctly identified malicious node is a true positive (TP). Our approach takes the order of minutes to analyze a large-sized enterprise's day long data, and identifies previously unknown malicious domains.

1.2 Contributions and Roadmap

We make the following contributions in this paper.

- We demonstrate that we can extract actionable security information from enterprise event logs in a scalable and reliable manner.
- We model the malicious domain detection problem as a graph inference problem and adapt belief propagation to solve the problem. Our approach does not require additional data beyond event logs, does not compute features, and uses minimal data from existing blacklists and whitelists.
- We apply our approach to event logs collected at a global enterprise over 7 months and show that our approach scales well and identifies new malicious domains not present in the blacklists.

The rest of the paper is organized as follows. We introduce our graph inference approach in Section 2. We describe our data set and analysis setup in Section 3. We present and discuss our experimental results in Section 4. We compare our work with related work in Section 5 and conclude with a discussion of future work in Section 6.

2 A Graph Inference Approach

In this section, we describe our approach to detect malicious domains. We assume that we have an enterprise's *host-domain graph*, i.e., we know the domains accessed by the enterprise's hosts. We construct the graph by adding a node for each host in the enterprise and each domain accessed by the hosts and then adding edges between a host and its accessed domains. We can construct the graph from multiple enterprise event log datasets, e.g., HTTP proxy logs and DNS request logs.

We also assume that we know a few nodes' *states*, e.g., malicious or benign. A domain present in a domain black list or a domain white list is known to be

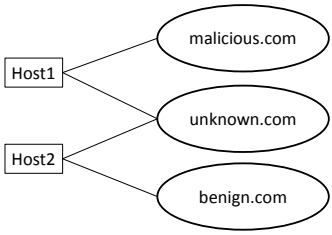

Fig. 1. A host-domain graph containing a malicious domain, *malicious.com*, a benign domain, *benign.com*, and an unknown domain, *unknown.com*

malicious or benign, respectively. Similarly, a malware infected host is known to be malicious whereas an uninfected host is known to be benign. The known nodes constitute our *ground truth* data set; the rest of the graph nodes are *unknown* nodes. We note that a small fraction of the graph's nodes are present in the ground truth data set. We show an example host-domain graph in Figure 1. The graph contains a malicious node, *malicious.com*, a benign node, *benign.com*, and three unknown nodes, *Host1*, *Host2*, and *unknown.com*.

Given the host-domain graph and the ground truth information, our goal is to *infer* the states of the unknown domains in the graph. For example, in Figure 1, we would like to infer the state of *unknown.com*. Formally, we would like to compute a node's *marginal probability* of being in a state, i.e., the probability of the node being in a state given the states of other nodes in the graph. We will then label the nodes with high marginal probability of being malicious as malicious nodes and benign otherwise.

In principle, our approach can detect both malicious domains and infected hosts in the enterprise. For example, in Figure 1, we can infer the states of *Host1* and *Host2*. In this paper, however, we focus on malicious domain detection.

Marginal probability estimation in graphs is known to be NP-complete [5]. Belief propagation (BP), introduced in the next section, is a fast and approximate technique to estimate marginal probabilities. BP's time complexity and space complexity are linear in the number of edges in a graph; hence BP scales well to large graphs. A typical enterprise host-domain graph has millions of nodes and tens of millions of edges. Hence we chose BP as our inference technique due to its scalability and its successful application in diverse fields such as computer vision [6], error correcting codes [7], fraud detection [8], and malware identification [9].

BP relies on ground truth information and *statistical dependencies* between neighboring nodes to reliably estimate marginal probabilities. The dependencies are derived from our domain knowledge. For example, user activities on benign hosts result primarily in benign domain accesses and occasional unintended malicious domain accesses, e.g., via phishing. Hence we may assume that benign hosts are more likely to visit benign domains than malicious domains. Similarly, a benign domain's neighbor is more likely to be a benign host than a malicious host. Malicious hosts may visit benign domains due to user activities; however, they are more likely to visit malicious domains as malware tend to contact many malicious domains. Intuitively, *Host 1* in Figure 1 is more likely to be malicious

as it has a malicious neighbor. Similarly, *Host 2* is more likely to be benign, and *unknown.com* is equally likely to be either.

2.1 Belief Propagation

In this section, we summarize the BP algorithm; please see Yedida et al. for details [5]. Judea Pearl introduced the BP algorithm for trees [4]. BP is an efficient technique to solve inference problems on graphical models. Given an undirected graph, $G = (V, E)$, where V is a set of n nodes and E is a set of edges, we model every node $i \in V$ as a random variable, x_i, that can be in one of a finite set, S, of states. A graphical model defines a joint probability distribution, $P(x_1, x_2, ..., x_n)$, over G's nodes. The inference process computes the marginal probability distribution, $P(x_i)$, for each random variable, x_i. A node's marginal probability is defined in terms of sums of the joint probability distribution over all possible states of all other nodes in the graph, i.e., $P(x_i) = \sum_{x_1} .. \sum_{x_{i-1}} \sum_{x_{i+1}} .. \sum_{x_n} P(x_1, x_2, ..., x_n)$. The number of terms in the sum is exponential in the number of nodes, n. BP, however, can approximate the marginal probability distributions of all nodes in time linear in the number of edges, which is at most $O(n^2)$.

BP estimates a node's marginal probability from prior knowledge about the graph's nodes and their statistical dependencies. A node, i's, belief, $b_i(x_i)$, is i's marginal probability of being in the state x_i. $b_i(x_i)$'s computation depends on *priors* of the graph nodes. A node, i's, prior, $\phi_i(x_i)$, is i's initial (or prior) probability of being in the state x_i. In our model, a node's priors indicate the node's initial likelihood of being in malicious and benign states. We estimate a node's priors using our ground truth information. $b_i(x_i)$'s computation also depends on *edge potential* functions that model the statistical dependencies among neighboring nodes. The edge potential, $\psi_{ij}(x_i, x_j)$, between two neighboring nodes, i and j, is the probability of i being in the state x_i and j being in the state x_j.

BP achieves computational efficiency by organizing global marginal probability computation in terms of smaller local computations at each node. This is done via *iterative* message passing among neighboring nodes. Consider a node, i, and its neighbors, $N(i)$. In each iteration of the algorithm, i passes a *message vector*, m_{ij}, to each of its neighbors, $j \in N(i)$. Each component, $m_{ij}(x_j)$, of the message vector is proportional to i's perception of j's likelihood of being in the state x_j. i's outgoing message vector to its neighbor j depends on i's incoming message vectors from its other neighbors and is computed as follows.

$$m_{ij}(x_j) = \sum_{x_i \in S} \phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus j} m_{ki}(x_i) \tag{1}$$

The order in which messages are passed is not important as long as all messages are passed in each iteration. Malware communication, e.g., bots communicating with C&C servers, provides a structural advantage in using BP as messages can propagate over multiple hops. In a *synchronous* update order, i's outgoing messages in iteration t is computed from i's incoming messages in

iteration $t-1$. In an *asynchronous* update order, incoming messages are used as soon as they are available. We chose to use a synchronous update order for its simplicity. The iterations stop when the messages converge within a small threshold, i.e., messages don't change significantly between iterations, or when a threshold number of iterations is reached. We then compute a node, i's, belief values from i's incoming messages in the converged or the last iteration.

$$b_i(x_i) = C\phi(x_i) \prod_{k \in N(i)} m_{ki}(x_i) \tag{2}$$

C is a normalization constant to ensure that i's beliefs add up to 1, i.e., $\sum_{x_i \in S} b_i(x_i) = 1$.

In the case of trees, BP always converges and the beliefs represent accurate marginal probabilities. But if a graph has loops, then belief propagation on the graph may not converge or may converge to inaccurate marginal probabilities [10]. In practice, however, belief propagation has been successful on graphs with loops: it converges quickly to reasonably accurate values [11].

3 HTTP Proxy Data Analysis

In this section, we describe our approach's application on an enterprise HTTP proxy data set. An HTTP proxy acts as an intermediary between an enterprise's hosts and the domains accessed by the hosts. Hence we can determine the domains visited by the hosts from proxy logs and construct an enterprise's host-domain graph.

3.1 Data Set and Graph Generation

We collected proxy logs over a 7 month period from August 2013 to February 2014 from 98 proxy servers in a global enterprise's worldwide locations. Each entry in the log represents an HTTP request and contains the requesting host's IP address, the domain requested, a time stamp, an HTTP header, and the request status. If we see an HTTP request from an IP address, I, for a domain, D, then we create two nodes, I and D, in the host-domain graph, and add an edge between I and D.

We follow standard terminology and note that given a domain, *www.hp.com (or www.hp.co.uk)*, *hp.com (or hp.co.uk)* is the *second-level domain (2LD)* and *com (or co.uk)* is the *top-level domain* (TLD). The TLD is also known as a *public suffix*. We use only 2LDs in our graph– collapsing domain nodes in this manner increases the number of paths in the graph, making paths between nodes more likely and hence information propagation between nodes more likely. Such a choice also reflects our assumption that usually 2LDs are responsible for their domain's and sub-domains' security. If a proxy log contains an IP address instead of a domain name as the destination, we add the IP address as a graph node.

We represent hosts by their IP addresses in our graph. Since IP addresses are transient in nature, a single host may be represented by multiple graph nodes.

Table 1. Data and graph description. Each row in the table represents a time period, and the columns show the time period, the number of events in the time period, the number of nodes and edges in the graph constructed from the events, the number of known malicious nodes and benign nodes in the graph, and the number of known nodes as a percentage of all graph nodes (B = billion, M = million, and K = thousand).

Time Period	Events	Nodes	Edges	Malicious Nodes	Benign Nodes	Ground truth (%)
01-16-2014	1.29B	2.80M	27.8M	21.6K	19.7K	1.45
01-17-2014	1.19B	2.58M	25.3M	21.5K	19.8K	1.60
01-18-2014	0.40B	0.80M	5.33M	10.8K	9.41K	2.51
01-19-2014	0.36B	0.70M	4.17M	10.7K	9.45K	2.88
01-20-2014	1.02B	2.46M	22.0M	21.4K	19.7K	1.67
01-21-2014	1.26B	2.81M	27.9M	21.6K	19.8K	1.47
01-22-2014	1.00B	2.35M	23.2M	21.3K	19.7K	1.73
1 Week	6.52B	10.5M	85.2M	104K	103K	1.98
3 Hours-1	0.20B	0.78M	6.95M	5.62K	4.66K	1.32
3 Hours-2	0.22B	0.76M	7.27M	5.80K	4.65K	1.38
6 Hours-1	0.31B	1.08M	8.90M	8.76K	7.65K	1.52
6 Hours-2	0.41B	1.21M	12.1M	9.06K	7.59K	1.38

We, however, observe that IP address assignment in enterprise networks is stable over periods of days and even months.

Table 1 shows the summary of one week's data collected from January 16^{th}, 2014 to January 22^{nd}, 2014. For each day (column 1), we show the number of log events (column 2, in billions), and the numbers of nodes (column 3, in millions) and edges (column 4, in millions) in the graph constructed from the day's logs. The 7^{th} row in the table shows the numbers for the graph constructed from the week's data. January 18^{th} and 19^{th} were weekend holidays; hence the numbers of events collected on those days are much less than the numbers on weekdays.

3.2 BP Parameters

We obtained blacklists of known malicious domains and IP addresses from a commercial blacklist and seventeen freely available lists including *OpenBL.org* and *malwaredomains.com* projects. Since domain blacklists change frequently, we obtained blacklists from the same time period as the logs. We use Alexa's popular domain list as our whitelist [12], where we chose top K entries to be benign domains to maintain a balance between malicious and benign domains. Table 1 shows malicious nodes (column 5, in thousands) and benign nodes (column 6, thousands) in each graph. These nodes represent our ground truth information; column 7 shows ground truth as a percentage of all graph nodes. Since our focus was on detecting malicious domains, we did not use any ground truth information for the host nodes.

We assign priors to graph nodes according to our ground truth data. For example, we assign a prior, $P(malicious) = 0.99$, to the nodes present in the

Table 2. Priors assigned to a node according to the node's state

Node	P(malicious)	P(benign)
Malicious	0.99	0.01
Benign	0.01	0.99
Unknown	0.5	0.5

blacklist. We do not assign a probability of 1 to account for possible errors in our ground truth data. Table 2 shows the prior assignments according to whether a node is known malicious, known benign, or unknown. We assume that an unknown node is equally likely to be malicious or benign.

We introduce an edge potential matrix to reflect the statistical dependencies among neighboring nodes. We assume a *homophilic* relationship, i.e., two neighboring nodes are more likely to be of the same state than different states. For example, a malicious host and a malicious domain are more likely to be neighbors than a benign host and a malicious domain. The relationship is based on our intuition that hosts that visit benign sites are likely to be benign and hosts that visit malicious sites are likely to be infected. Table 3(a) shows our edge potential matrix. We explore more parameter choices in Section 4.2.

Table 3. Edge potential matrices

(a)

x_i \ x_j	Benign	Malicious
Benign	0.51	0.49
Malicious	0.49	0.51

(b)

x_i \ x_j	Benign	Malicious
Benign	0.75	0.25
Malicious	0.49	0.51

3.3 Experimental Setup

We implemented the BP algorithm in `Java` and ran our experiments on a 12-core 2.67 GHz desktop with 96GB of RAM. Since our graph has many high degree nodes, e.g., degree $>$ 100K, and the incoming messages are less than 1, multiplying all incoming messages results in underflow, i.e., multiplication results in 0. We handled underflow in two ways. First, we used `Java's` *BigDecimal* data type to perform arbitrary precision operations; we, however, pay a performance penalty. Second, we *normalize* outgoing message vectors, i.e., we ensure that a vector's components add up to 1. For example, instead of sending a vector, (0.0023, 0.0023), we normalize the vector to (0.5, 0.5). The larger normalized numbers help avoid underflow.

We constructed our graphs off-line, i.e., we stored the logs in compressed format on disk and then uncompressed them in memory to create the graphs. Graph construction from a week day's compressed data took an average of 5

hours with peak memory usage of 9GB. In practice, however, graph construction will be done online: as and when event logs are generated, new nodes and edges will be added to the graph as needed. Hence at any point in time, the day's graph will be up-to-date. If we need to store historical data, we can store the graphs and not the event logs. A weekday's graph requires an average of 426MB of disk space.

The peak memory usage during BP's iteration phase was 53GB. The average iteration time was 7.8 minutes on a weekday's data and 1.25 minutes on a weekend day's data. The BigDecimal data type's use was a major contributor to iteration time and memory usage.

We used *message damping* to speed up convergence [13]. If m_{ij}^{t-1} is the outgoing message from a node, i, to its neighbor, j, in $t-1^{th}$ iteration, then the outgoing message in the t^{th} iteration is $m_{ij}^{t} = \alpha m_{ij}^{t-1} + (1-\alpha)\bar{m}_{ij}^{t}$, where $\bar{m}_{ij}^{t}$ is the outgoing message in the t^{th} iteration as computed by Equation 1 and α is a damping factor in the range [0,1]. We experimented with a range of values for α and empirically determined that $\alpha = 0.7$ produces the best performance.

We ran each experiment till either BP converged or 15 iterations were completed. We then computed the belief values as defined in Equation 2.

3.4 Result Computation

Following standard practice, we use *K-fold cross validation* to compute our malicious domain detection performance, i.e., we divide the ground truth data into K folds, mark one fold as *test* data and the remaining *K-1* folds as *training* data. We seed the host-domain graph with the training data, reset the priors of the nodes in the test data to unknown priors, run belief propagation, and then compute beliefs following the procedure described in the previous subsections. We then compute our detection performance on the test fold. We repeat the process for each of the K folds and report our average performance over the K folds. We describe the process of selecting K in the next section.

We present our malicious domain detection results as Receiver Operating Characteristics (ROC) plots, i.e., plots showing false positive rates and true positive rates. Since low FPRs are essential in enterprise settings, we chose ROC plots instead of overall classification accuracy. We obtain an ROC plot by *thresholding* a node's malicious belief value. For example, given a threshold, t, if a node, n's, malicious belief, $b_n(malicious) > t$, then we predict n as malicious; else, n is benign. We then use n's ground truth state and predicted state to label n as false positive, true positive, false negative, or true negative. For example, given a malicious node, n, in ground truth, if we predict n as malicious, then n is a true positive; else n is a false negative. Similarly, given a benign node, n, in ground truth, if we predict n as benign, then n is a true negative; else n is a false positive. We then repeat the process for all nodes in the test fold to compute the FPR and the TPR at threshold t. We then vary t uniformly in the range [0,1] to obtain an ROC plot. Network administrators can pick an operating point on the plot according to their risk profiles. For example, they may choose a high

threshold to reduce FPs or a low threshold to increase detection at the risk of increasing FPs.

4 Results and Discussion

In this section, we first present our experiments on BP's parameter selection and then present our malicious domain detection results. We also demonstrate our ability to detect new malicious domains that are unlikely to be present in externally sourced blacklists.

4.1 K-Fold Cross Validation

We conducted an experiment to compare our performance under 3 different values of K: 2, 3, and 10. We constructed a host-domain graph from January 16^{th}'s event logs, seeded the graph using the day's blacklist and whitelist, used BP parameters described in Section 3.2, and then performed K-fold cross validation. We show the ROC plots in Figure 2. The areas under the ROC curves (AUC) are 98.53%, 98.72%, and 98.80% for $K = 2$, 3, and 10, respectively. The higher the AUC, the better the classification result. Hence, $K = 10$ produces the best classification result. Also, 10-fold cross validation is the standard practice in classification tasks. Hence we use $K = 10$ in our subsequent experiments.

When $K = 10$, we use $9/10^{th}$ of the ground truth data as training data whereas we use only half the ground truth data as training data when $K = 2$. Hence the result confirms our intuition that everything else being equal, more training data leads to better detection results.

4.2 Parameter Sensitivity Analysis

We conducted parameter sensitivity analysis on January 16^{th}'s graph to study priors' and edge potentials' impact on our results. We experimented with different priors values, e.g., $\{0.95, 0.05\}$ for malicious nodes, instead of the values shown in Section 3.2 for known nodes; our performance did not change. We also assigned priors to unknown nodes according to the nodes' attributes. For unknown domains, we assume that popular domains are likely to be benign. Hence we assign a *sigmoid* function, $1/(1 + \exp(-d))$, of an unknown domain node's degree, d, as the benign prior. We also assume that malware infected hosts make large number of HTTP requests, e.g., bots trying to contact their command and control server. Hence we assign a sigmoid function of an unknown host node's HTTP request count as the malicious prior. The sigmoid prior's ROC plot is marginally inferior to Figure 2; hence we omit the plot due to space limitation.

We also experimented with the edge potential matrix shown in Table 3(b); we assume the prevalence of beneficence and assign a lower probability to spread of malware. Our FPR and TPR does not change from Figure 2. Hence our approach is robust with respect to our parameter choices and we use the parameters shown in Section 3.2 for our subsequent experiments.

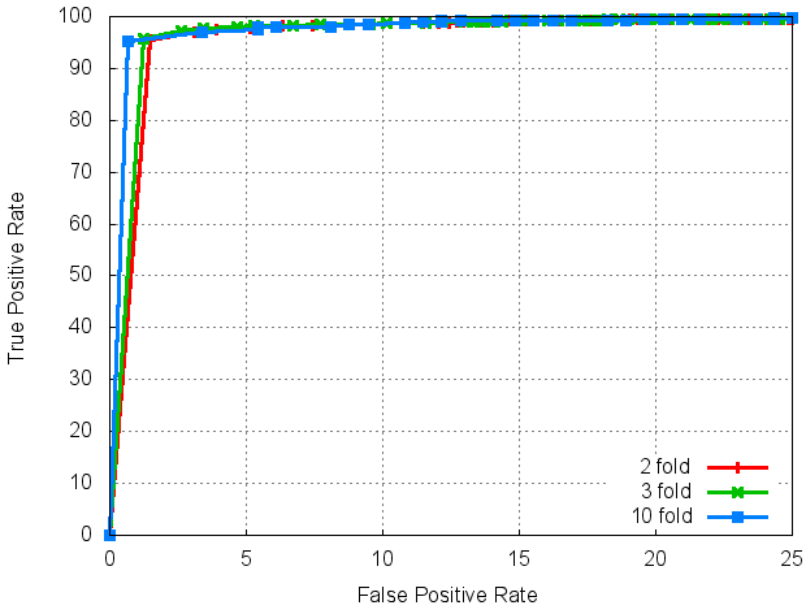

Fig. 2. ROC plots for different K-fold cross validations. For clarity, the X-axis ends at FPR = 25%. $K = 10$ performs the best.

4.3 Malicious Domain Detection

We present our malicious domain detection results on data collected on 7 consecutive days in January 2014. For each day, we constructed a host-domain graph from the day's logs, seeded the graph from the day's whitelist and blacklist, used parameters described in Section 3.2, and then performed 10-fold cross validation. We show the ROC plots in Figure 3.

The plots show that our approach can achieve high detection rates with low false positive rates using minimal ground truth information. For example, on January 16^{th}, the host-domain graph has only 1.45% nodes in the ground truth data; yet we achieve a 95.2% TPR with a 0.68% FPR.

The results obtained on weekdays' data are similar to that of January 16^{th}. The results on weekends, however, are inferior. For example, on January 18^{th}, we achieve a 96.7% TPR at a high FPR of 5.6%. The AUC on January 16^{th} is 98.80% compared to 96.12% on January 18^{th}.

The number of events on the weekend days is smaller than the weekdays due to less activity in the enterprise over the weekend; hence the graphs on weekend days have fewer edges (Table 1). January 18^{th}'s average domain degree is 8.47 compared to 11.45 on January 16^{th}. Hence the weekend's graphs do not include complete node behavior and have fewer paths in the graphs for information propagation. These two reasons may have caused the inferior performance.

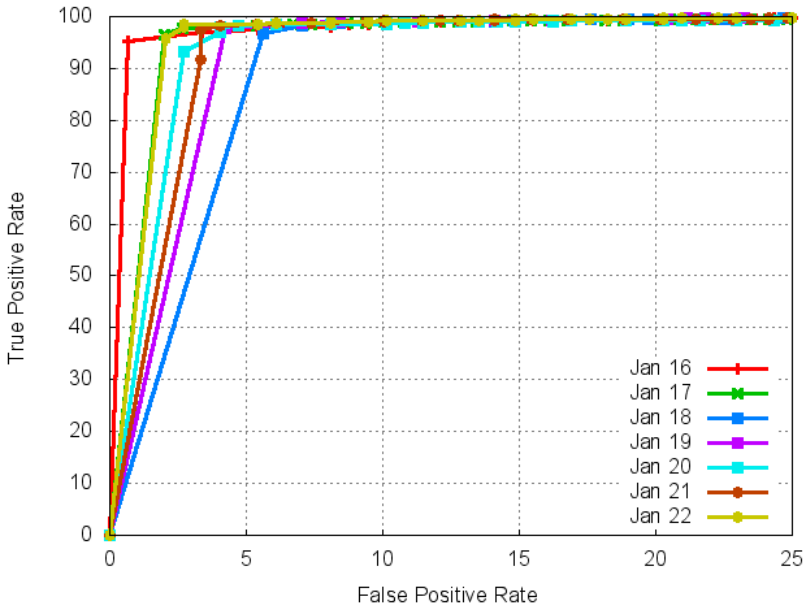

Fig. 3. ROC plots for 7 days in January 2014. Our approach performs better over weekdays than weekend days.

4.4 More Event Logs

Given the discussion above, we naturally examined the question of whether more event logs collected over a time period longer than a day, leads to better performance. Intuitively, we believed that more event logs will capture more complete node behavior. Hence we constructed a single host-domain graph from the logs collected over the 7 days in January. We also combined the blacklists and whitelists over the 7 days to a single whitelist and a single blacklist. Figure 4 shows the ROC plot for the graph and the combined whitelist and blacklist. The results are counterintuitive.

Our performance on 7 days data is inferior to its performance on a single day's data. For example, we achieve a TPR of only 90.2% at the FPR of 3.09%. Two key reasons contribute to the poor performance. First, the combined graph's average domain degree is less than those of the graphs constructed from single days' data. This is due to the fact that the enterprise's hosts visited many new domain nodes every day; these domains were not present in the previous days' logs. Second, combining the blacklists and whitelists may have introduced errors in the ground truth information. For example, a domain may have been malicious the first day and might have been cleaned up later. But our approach will consider the domain as malicious for the entire 7 day period.

Furthermore, the average iteration time on the combined graph was 31.0 minutes. Hence we do not recommend our approach over longer time scale data.

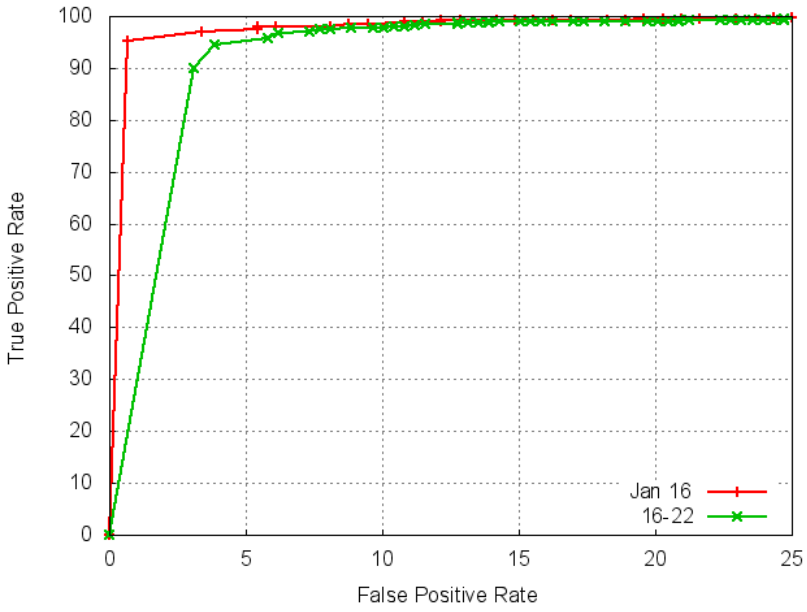

Fig. 4. Our performance on a week's data is inferior to a single day's data

4.5 Detection Details

In this section, we examine our false positives and true positives. Most of our FPs, i.e., benign domains identified as malicious by our approach, are of low degree. Though these domains are in Alexa's popular list and are globally popular, e.g., an Indian matrimonial site, very few hosts in the enterprise access them. If an administrator blocks access to such domains, they will not impact business activities. However, blocking access to popular domains such as *shopping.hp.com* and *google.com* will be catastrophic. Our approach didn't commit any such mistake in our experiments.

We also examine new malicious domains identified by our approach. These are not present in the blacklist and hence are unknown domains in the host-domain graph. BP assigns high malicious beliefs to these domains and classifies them as malicious. We show a few such domains in Table 4.

Table 4. Our approach identifies new malicious domains

luo41cxjsbxfrhtbxfubxaqawhxjshsjx.info
awhvkvkzk17fxa67e51pvp42ozmyiqhvfwp12.info
etn30aqjxf12e61d30hxkxhxgvktmqaqkqdu.info
f32pxntk37gxgxmqn30bzhqpqavovbqgtk67.ru

These random looking domain names are likely to be algorithmically generated by malware resident on the enterprise's hosts. For example, bots typically generate new domains every day to contact their command and control server. Externally sourced blacklists are unlikely to contain these domains for three reasons. First, the list generation process may not be aware of the domains; even if the process had access to malware, the malware may generate different domains every time it runs. Second, the domains are active for a short time period, e.g., a few hours, and might not exist by the time they are added to lists. Third, there are many such domains and adding all of them will increase the list size without much benefit.

Our results show that we can take advantage of the externally sourced blacklists and identify previously unknown malicious domains not present in the lists.

4.6 Near Real Time Detection

Our system implementation's average completion time for 15 iterations on a day's data was 115 minutes. Hence, we can construct the host-domain graph online and run our approach every 115 minutes. In the best case, our approach can detect a domain 115 minutes after the domain's first access. This delay might be unacceptable in some sensitive settings. Hence we experimented with smaller data sets. We divided January 16th's data into 3 hours and 6 hours blocks, constructed a host-domain graph from each block, and then applied BP. Due to space limitation, we show a few representative graphs in Table 1's last 4 rows and ROC plots in Figure 5. Our detection performance on smaller datasets is marginally inferior to a day's data. The time gain, however, is compelling. For example, 15 iterations took 16.6 minutes for completion on 3 hours' data and 37.5 minutes for 6 hours' data. Hence in principle, our approach can run every 17 minutes and detect previously unknown domains.

4.7 Seven Months' Data

Finally, we demonstrate that our detection results over 7 days in January 2014 are not due to extraneous reasons. We randomly chose 7 days from the 7 months, one day from each month, and applied BP on the host-domain graphs obtained from each day's event logs. Figure 6 shows the ROC plots. The plots are similar to the plots obtained from data collected in the days in January.

5 Related Work

In this section, we compare our work with related work in big data analysis for security and malicious domain detection. Yen et al. analyze HTTP proxy logs to identify suspicious host activities– they extract features from the logs and then use clustering to find outlying suspicious activities [14]. Their approach, though carried on smaller scale data, is complimentary to ours; they focus on host activity detection and we focus on malicious domain detection. Giura et al.

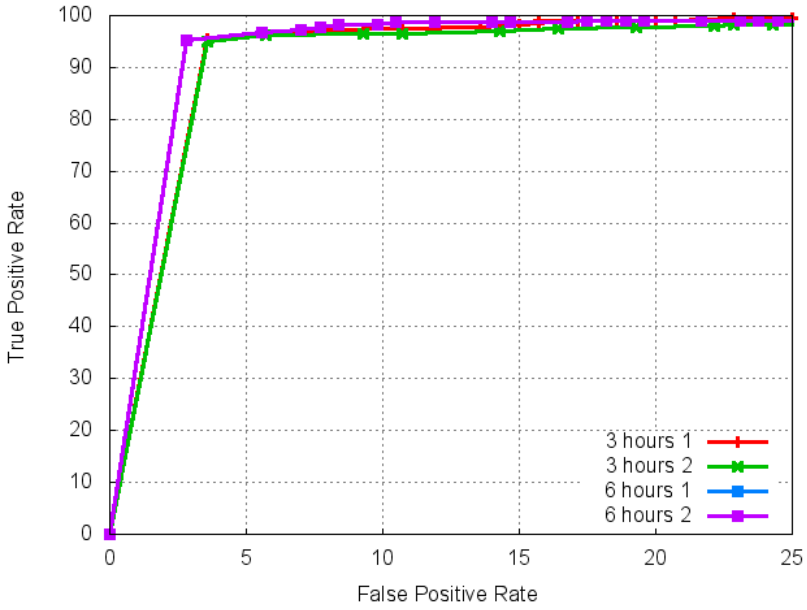

Fig. 5. Few ROC plots for 3 hours' and 6 hours' data. The results are marginally inferior to a day's data.

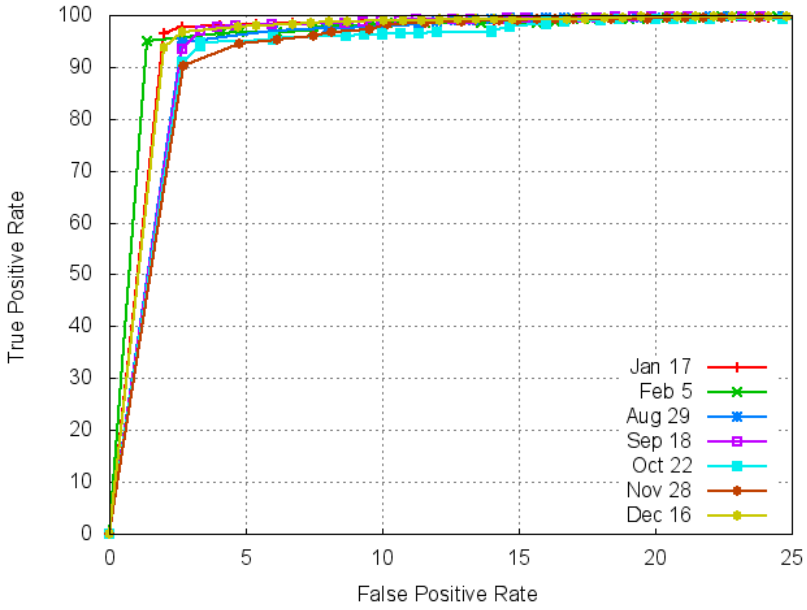

Fig. 6. ROC plots for 7 days' data, one randomly chosen from each of the 7 months, are similar to Figure 3

propose an attack pyramid model to identify advanced persistent threats from network events [15]. Bilge et al. analyze netflow data to identify botnet command and control servers [16].

Multiple malicious domain detection approaches have been proposed. Yadav et al. detect fast flux domains from DNS traffic by looking for patterns in algorithmically generated domain names [17]. EXPOSURE [18], Kopis [19], and Notos [20] use passive DNS analysis to detect malicious domains: they compute multiple features for domains names, and then perform automated classification and clustering using training data. For example, Notos uses network features, zone features, features related to whether domain names were discovered by a honeypot and whether domain names were present in black lists. EXPOSURE uses features based on time of DNS queries, answers, time-to-live (TTL) values, and domain name syntax. Failed DNS queries have also been analyzed to detect malicious domains. Antonakakis et al. use a combination of clustering and classification of failed DNS queries to detect malware generated domains names [21]. Jiang et al. construct a DNS failure graph, extract dense subgraphs, and show that the subgraphs represent anomalous activities such as bots [22]. Yadav et al. use DNS failures' temporal and entropy based features to detect C&C servers [23]. Our work differs from these works in the following aspects: we use event logs routinely collected by enterprises; we require no additional data collection, whether passive data, e.g., zone information, or active data, e.g., honeypot interaction data. Also, extensive feature computation may be prohibitive in large enterprise settings. Hence our approach requires no feature computation and uses minimal training data.

Multiple malicious URL identification approaches have also been proposed. Anderson et al. use clustering by graphical similarity to detect spam URLs [24]. Lin et al. introduce a lightweight approach to filter malicious URLs by using lexical and descriptive features extracted from URL strings [25]. Ma et al. introduce an URL classification system by using statistical methods to discover lexical and host-based properties of malicious URLs [26]. Thomas et al. use logistic regression on extracted features to determine if an URL directs to spam content [27]. Zhang et al. use lexical features and term frequency/inverse document frequency algorithm to detect phishing URLs [28]. These approaches classify individual URLs, e.g., *maldom.com/url1*, as malicious whereas our approach identifies an entire domain, e.g., *maldom.com*, as malicious and hence labels all associated URLs with the domain, e.g., *maldom.com/url**, as malicious.

6 Summary and Future Work

In this paper, we introduced a graph inference approach for detecting malicious domains accessed by an enterprise's hosts. Our experiments on seven months' of HTTP proxy data collected at a global enterprise show that belief propagation is a reliable and scalable approach and can detect previously unknown malicious domains. Our work is an example of big data analysis for security, i.e., analyzing enterprise event data to extract actionable security information. In the future,

we plan to extend our work to track the spread of malware infections inside an enterprise network. We also plan to explore big data analysis approaches for other types of enterprise event logs such as DNS logs and firewall logs.

Acknowledgments. The authors thank Marc Eisenbarth, Stuart Haber, and A. L. Narasimha Reddy for helpful discussions and feedback at various stages of the research.

References

1. CRA: Challenges and opportunities with big data (2012), http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf
2. Cardenas, A.A., Manadhata, P.K., Rajan, S.P.: Big data analytics for security. IEEE Security & Privacy 11(6), 74–76 (2013)
3. Symantec internet security threat report (2011), http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf
4. Pearl, J.: Reverend bayes on inference engines: a distributed hierarchical approach. In: Proceedings of the National Conference on Artificial Intelligence (1982)
5. Yedida, J., Freeman, W., Weiss, Y.: Understanding Belief Propagation and its Generalizations. Exploring Aritificial Intelligence in the New Millennium (2003)
6. Freeman, W.T., Pasztor, E.C., Carmichael, O.T.: Learning low-level vision. International Journal of Computer Vision 40(1), 25–47 (2000)
7. Mceliece, R., Mackay, D., Cheng, J.: Turbo decoding as an instance of pearl's belief propagation algorithm. IEEE Journal on Selected Areas in Communications (1998)
8. Pandit, S., Chau, D.H., Wang, S., Faloutsos, C.: Netprobe: a fast and scalable system for fraud detection in online auction networks. In: World Wide Web Conference (2007)
9. Chau, D., Nachenberg, C., Wilhelm, J., Wright, A., Faloutsos, C.: Polonium: Terascale graph mining and inference for malware detection. In: SIAM International Conference on Data Mining (2011)
10. Murphy, K., Weiss, Y., Jordan, M.: Loopy Belief Propagation for Approximate Inference: An Empirical Study. Uncertainity in Artificial Intelligence (1999)
11. Frey, B.J., MacKay, D.J.C.: A revolution: Belief propagation in graphs with cycles. In: Neural Information Processing Systems (NIPS) (1997)
12. Alexa: Top Sites, http://www.alexa.com/topsites
13. Pretti, M.: A message-passing algorithm with damping. Journal of Statistical Mechanics: Theory and Experiment 2005(11), P11008 (2005)
14. Yen, T.F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., Kirda, E.: Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In: Proceedings of the 29th Annual Computer Security Applications Conference, ACSAC 2013, pp. 199–208. ACM, New York (2013)
15. Giura, P., Wang, W.: A context-based detection framework for advanced persistent threats. In: International Conference on Cyber Security (2012)
16. Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., Kruegel, C.: Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In: Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC 2012, pp. 129–138. ACM, New York (2012)

17. Yadav, S., Reddy, A.K.K., Reddy, A.N., Ranjan, S.: Detecting algorithmically generated malicious domain names. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, IMC 2010. ACM, New York (2010)
18. Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: EXPOSURE: Finding Malicious Domain Using Passive DNS Analysis. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2011)
19. Antonakakis, M., Perdisci, R., Lee, W., Vasiloglou II, N., Dagon, D.: Detecting malware domains at the upper dns hierarchy. In: 20th USENIX Security Symposium (2011)
20. Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., Feamster, N.: Building a Dynamic Reputation System for DNS. In: USENIX Security Symposium (2010)
21. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., Dagon, D.: From throw-away traffic to bots: Detecting the rise of dga-based malware. In: 21st USENIX Security Symposium (2012)
22. Jiang, N., Cao, J., Jin, Y., Li, L.E., Zhang, Z.L.: Identifying Suspicious Activities Through DNS Failure Graph Analysis. In: IEEE Conference on Network Protocols (2010)
23. Yadav, S., Reddy, A.L.N.: Winning with DNS failures: Strategies for faster botnet detection. In: Rajarajan, M., Piper, F., Wang, H., Kesidis, G. (eds.) SecureComm 2011. LNICST, vol. 96, pp. 446–459. Springer, Heidelberg (2012)
24. Anderson, D.S., Fleizach, C., Savage, S., Voelker, G.M.: Spamscatter: Characterizing internet scam hosting infrastructure. In: 16th USENIX Security Symposium (2007)
25. Lin, M., Chiu, C., Lee, Y., Pao, H.: Malicious URL filtering- a big data application. In: IEEE BigData (2013)
26. Ma, J., Saul, L.K., Savage, S., Voelker, G.M.: Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. In: Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) (June 2009)
27. Thomas, K., Grier, C., Ma, J., Paxson, V., Song, D.: Design and Evaluation of a Real-Time URL Spam Filtering Service. IEEE Security and Privacy (2011)
28. Zhang, Y., Hong, J., Cranor, L.: Cantina: A content-based approach to detecting phishing web sites. In: World Wide Web Conference (May 2007)