Detection of Botnets Using Combined Host- and Network-Level Information

Yuanyuan Zeng, Xin Hu, Kang G. Shin
{gracez,huxin,kgshin}@eecs.umich.edu
University of Michigan, Ann Arbor, MI 48109-2121, USA

Abstract

Bots are coordinated by a command and control (C&C) infrastructure to launch such attacks as Distributed-Denial-of-Service (DDoS), spamming, identity theft and phishing, all of which seriously threaten the Internet services and users. Most contemporary botnet-detection approaches have been designed to function at the network level, requiring the analysis of packets' payloads. However, analyzing packets' payloads raises privacy concerns and incurs large computational overheads. Moreover, network traffic analysis alone can seldom provide a complete picture of botnets' behavior. By contrast, general in-host detection approaches are useful to identify each bot's host-wide behavior, but are susceptible to the host-resident malware if used alone. To address these limitations, we account for both the coordination within a botnet and the malicious behavior each bot exhibits at the host level, and propose a C&C protocol-independent detection framework that combines both host- and network-level information for making detection decisions. This framework clusters similarly-behaving hosts into groups based on network-flow analysis without accessing packets' payloads, and then correlates the clusters with each individual's in-host behavior for validation. The framework is shown to be effective and incurs low false-alarm rates in detecting various types of botnets.

1 Introduction

Botnets have now become one of the most serious security threats to Internet services and applications. A *bot* is a computer compromised by worms, Trojan horses, or backdoors, under a remote command and control (C&C) infrastructure. A group of coordinated bots is called a *botnet*, and can cooperatively mount Distributed-Denial-of-Service (DDoS) attacks, spamming, phishing, identity theft, and other cyber crimes.

To control a botnet, a botmaster needs to use a C&C channel to issue commands, and coordinate bots' actions. Traditional botnets utilize the IRC protocol as their C&C infrastructure. Attackers set up an IRC server and specify a channel via which bots connect to, and listen on, in order to receive commands from botmasters. HTTP-based botnets are similar to the IRC-based ones, but after infection, bots contact a web-based C&C server and notify the server with their system-identifying information via HTTP. This server sends back commands via HTTP responses. Although IRC- and HTTP-based C&C have been adopted by many past and current botnets, both of them are vulnerable to a central-point-of-failure. That is, once the central IRC or HTTP server is identified and removed, the entire botnet will be disabled.

To counter this weakness, attackers have recently shifted toward a new generation of botnets utilizing decentralized C&C protocols such as P2P. This C&C infrastructure makes detection and mitigation much harder. A well-known example is the Storm worm (a.k.a. Nuwar, W32.Peacomm, and Zhelatin) [4] which spreads via email spam and is known to be the first malware to seed a botnet in a hybrid P2P fashion. Storm uses peers as HTTP proxies to relay C&C traffic and hides the botmasters well behind the P2P network. Storm was estimated to run on between 250,000 and 1 million compromised systems in 2007. The Storm botnet has been used in some criminal activities, primarily for sending spam emails. A recent spambot Waledac, which came to the wild at the end of 2008, also spreads via spam emails and forms its botnet using a C&C structure similar to the Storm botnet. Some researchers pointed out that Waledac is the new and improved version of the Storm botnet [18].

To date, most botnet-detection approaches operate at the network level; a majority of them target traditional IRC- or HTTP-based botnets [12, 5, 10, 14, 17, 22] by looking for traffic signatures or flow patterns. We are aware of only one approach [11] designed for protocol- and structure-independent botnet detection. This approach requires packet-level inspection and depends solely on network traffic analysis unlikely to have a complete view of botnets' behavior. We thus need the finer-grained host-by-host behavior inspection to complement the network analysis. On the other hand, since bots behave maliciously system-wide, general host-based detection can be useful. One such way is to match malware signatures, but it is effective in detecting known bots only. To deal with unknown bot infiltration, in-host behavior analysis [6, 15, 8, 21, 20] is needed. However, since some in-host malicious behavior is not exclusive to bots and in-host mechanisms are vulnerable to host-resident malware, host-based approaches alone can hardly provide reliable detection results and thus we need external, hard-to-compromise (i.e., network-level) information for more accurate detection of bots' malicious behavior.

Considering the required coordination within each botnet at the network level and the malicious behavior each bot exhibits at the host level, we propose a C&C protocol-independent detection framework that incorporates information collected at

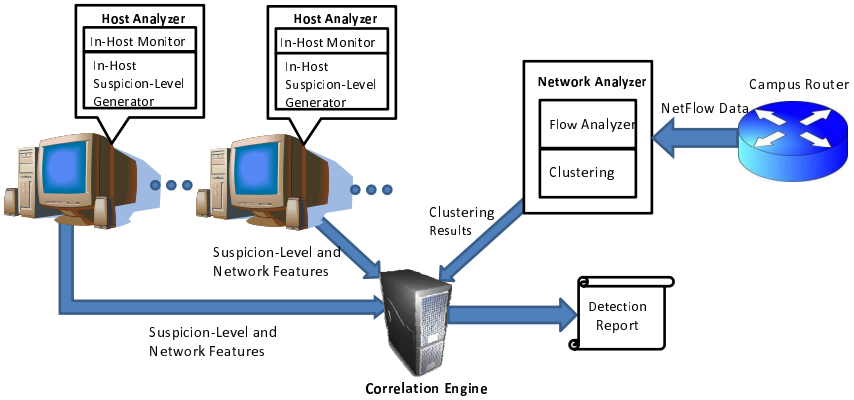

Figure 1. System architecture

both the host and the network levels. The two sources of information complement each other in making detection decisions. Our framework first identifies suspicious hosts by discovering similar behaviors among different hosts using network-flow analysis, and validates the identified suspects to be malicious or not by scrutinizing their in-host behavior. Since bots within the same botnet are likely to receive the same input from the botmaster and take similar actions, whereas benign hosts rarely demonstrate such correlated behavior, our framework looks for flows with similar patterns and labels them as triggering flows. It then associates all subsequent flows with each triggering flow on a host-by-host basis, checking the similarity among those associated groups. If multiple hosts behave similarly in the trigger-action patterns, they are grouped into the same suspicious cluster as likely to belong to the same botnet. Whenever a group of hosts are identified as suspicious by the network analysis, the host-behavior analysis results based on a history of monitored host behaviors are reported. A correlation algorithm finally assigns a detection score to each host under inspection by considering both network and host behaviors.

Our contributions are three-fold. First, to the best of our knowledge, this is the first framework that combines both network- and host-level information to detect botnets. The benefit is that it completes a detection picture by considering not only the coordination behavior intrinsic to each botnet but also each bot's in-host behavior. For example, it can detect botnets that appear stealthy in network activities with the assistance of host-level information. Moreover, we extract features from NetFlow data to analyze the similarity or dissimilarity of network behavior without inspecting each packet's payload, thus preserving privacy. Second, our detection relies on the invariant properties of botnets' network and host behaviors, which are independent of the underlying C&C protocol. It can detect both traditional IRC and HTTP, as well as recent hybrid P2P botnets. Third, our approach was evaluated by using several days of real-world NetFlow data from a core router of a major campus network containing benign and botnet traces, as well as multiple benign and botnet data sets collected from virtual machines and regular hosts. Our evaluation results show that the proposed framework can detect different types of botnets with low false-alarm rates.

The remainder of the paper is organized as follows. Section 2 provides an overview of our system architecture. Section 3 details the proposed detection methodology. Implementation and evaluation results are presented in Section 4 and 5. Limitations are discussed in Section 6. Section 7 describes the related work. The paper concludes with Section 8.

2 System Architecture

Figure 1 shows the architecture of our system, which primarily consists of three components: host analyzer, network analyzer, and correlation engine.

As almost all of current botnets target Windows machines, our host analyzer is designed and implemented for Windows platforms. The host analyzer is deployed at each host and contains two modules: in-host monitor and suspicion-level generator. The former monitors run-time system-wide behavior taking place in the Registry, file system, and network stack on a host. The latter generates a suspicion-level by applying a machine-learning algorithm based on the behavior reported at each time window and computes the overall suspicion-level using a moving average algorithm. The host analyzer sends the average suspicion-level along with a few network feature statistics to the correlation engine, if required. The network analyzer also contains two modules: flow analyzer and clustering.

The flow analyzer takes the flow data from a router as input and searches for trigger-action botnet-like flow patterns among different hosts. It then extracts a set of features that can best represent those associated flows and transforms them into feature vectors. Those vectors are then fed to the clustering module that groups similarly-behaving hosts into the same cluster, assuming them likely to be part of a botnet. Whenever a suspicious group of hosts are identified by the network analyzer, their host analyzers are required to provide the suspicion-level and network statistics to the correlation engine, which verifies the validity of the host information by comparing the network statistics collected from the network and those received from the host. The correlation engine finally assigns a detection score to each host and produces a detection result.

3 Methodology

As described earlier, our framework consists of three main components: host analyzer, network analyzer, and correlation engine. Each of these components is detailed next.

3.1 Host Analyzer

The host-analyzer is composed of two modules: in-host monitor and in-host suspicion-level generator.

3.1.1 In-Host Monitor

Each in-host monitor captures system-wide behavior in real time at different locations. Before deploying monitors, we need to decide which behavior features to monitor. By studying contemporary bots' behaviors, we have observed that they share certain behavior patterns that are different from benign applications, and that their behaviors can be grouped into 3 categories taking place at the Registry, file system and network stack. For example, when infecting a computer, a bot first creates an exe or dll file in the system directory. It then registers an autorun key in the Registry to make itself run automatically whenever the host system boots up. It also injects its code into other processes to hide its presence and disables anti-virus software and the task manager, if necessary. Finally, it opens one or more ports for further communications and establishes connections with the botmaster or peers in order to launch DDoS, spamming activities, etc. Note that a single activity mentioned above may not be malicious because it is also likely to be performed by benign hosts. However, the combination and aggregation of these activities can reveal that a host has been infected, since chances are slim that a benign host conducts all of these activities. Thus, the in-host suspicion-level analysis considers the behavior features altogether while making decisions.

To facilitate a further analysis, each host's run-time behavior is transformed into a uniform format known as a *behavior vector*. Each behavior vector consists of 9 behavior features as shown in table 1. As mentioned earlier, these features are intrinsic to bot-infected hosts. For example, in the file system, a bot always drops its payload into the system directory because normal users seldom inspect this directory and the payload is less noticeable among thousands of system files. Based on this observation, we closely monitor the create and write accesses in the system directory. In the Registry, almost all bots will add a key to automatically run themselves when Windows starts up, and some inject themselves into other processes or modify critical Registry keys. We are therefore interested in capturing these typical Registry activities as well. The number of ports opened is of interest because bots always open new ports for communication. The number of suspicious ports provides a hint of malicious activities such as scanning or exploiting vulnerabilities in LSASS service at port 139 and RPC-DCOM at 445. As each host's network activities can be captured and analyzed at the network level, the in-host monitor should focus on behaviors that can't be observed externally, such as file and Registry operations, to complement the network-level information. However, since a host is vulnerable to being compromised, we need some information that can be obtained both internally and externally to validate the integrity of the data provided by a host. As a result, we have added a few network features (feature 7 to 9) for in-host monitoring; these features will be compared against the same features generated by the network-level analyzer.

Table 1. In-host behavior features

Index	Behavior Features
1	DLL or EXE Creation into System Directory
2	Modification of Files in System Directory
3	Creation of AutoRun Key in Registry
4	Creation of Process Injection Key in Registry
5	Modification of Critical Registry Key (Disabling taskmgr; Overriding antivirus, etc.)
6	Number of Ports Opened
7	Number of Suspicious Ports
8	Number of Unique IPs Contacted
9	Number of SMTP Flows

3.1.2 In-Host Suspicion-Level Generator

Given each host's behavior vector, we employ a supervised learning algorithm, or the *support vector machine* (SVM), to quantify its suspicion level. SVM learns from benign and malicious host behavior profiles prior to predicting unlabeled behavior vectors. Based on the training data, the SVM creates a hyperplane corresponding to a classification rule. Given a new behavior vector, the SVM estimates the distance of the sample from the hyperplane and decides which class it belongs to.

To make the most of this learning model, we calibrate the distance score to a posterior classification probability indicating how likely a test behavior vector belongs to a particular class [16]. The posterior probability is then translated into the suspicion level in [0, 1] where 0 is benign and 1 is bot-infected. The higher the suspicion level, the more likely it is bot-infected.

Since the suspicion level for each host is generated every time window, a bot may intentionally reduce its suspicion level by spreading malicious activities into different time windows or even sleeping for a while. To counter such an evasion attempt, we selectively accumulate the value in each field of the behavior feature vector. The features worth accumulation are those typical to bot-infected hosts, such as creating an autorun key in the Registry or injecting a piece of code into another process. In addition, we use the Exponential Weighted Moving Average (EWMA) algorithm to compute the average suspicion level every time window.

If Y_n denotes the suspicion level generated in the n-th time window, and S_{n-1} is the estimated average suspicion level at the $(n-1)$-th window, the estimated average at the n-th window is given by $S_n = \alpha * Y_n + (1-\alpha) * S_{n-1}$ where α is a constant smoothing factor. We define α as a function of the time interval between two suspicion-level readings. $\alpha = 1 - e^{-\frac{t_n - t_{n-1}}{W}}$ where $t_n - t_{n-1}$ is the length of the

time window of generating suspicion levels and W is the the period of time over which the suspicion level is averaged. The moving average can be expressed as

$$S_n = (1 - e^{-\frac{t_n - t_{n-1}}{W}}) * Y_n + e^{-\frac{t_n - t_{n-1}}{W}} * S_{n-1}. \quad (1)$$

Note that there are other weighted moving average algorithms available, but we use EWMA because it is reasonable to give a higher weight to recent observations and still not to ignore older observations.

3.2 Network Analyzer

Considering privacy concerns and computational cost, our network analyzer, which operates on the network traffic collected from a core router in a major campus network, only requires analysis of NetFlow data without accessing packets' payload. NetFlow is a network protocol developed by Cisco for summarizing IP traffic information [1]. A flow is defined as a sequence of packets between a source and a destination within a single session or connection. A NetFlow record contains a variety of flow-level information, such as protocol, source/destination IP and port, start and end timestamps, number of packets, and flow size, but has no packet content information. The network analyzer takes flow records from the router as input and generates host-clustering results. It consists of two modules: flow analyzer and clustering.

3.2.1 Flow Analyzer

The flow analyzer processes the flow records of all hosts in a network to extract trigger-action patterns of interest. Recall that bots within the same botnet usually receive the same input from botmasters and take similar actions thereafter. Such coordinated behaviors are essential and invariant to all types of botnets regardless of their C&C structure.

The first step in flow processing is to filter out irrelevant flows including internal flows and legitimate flows. Internal flows represent traffic within a network. Legitimate flows are those with well-known destination addresses such as Google and CNN which seldom function as C&C servers. Note that flow filtering is just an optional operation and not essential to our network analyzer. It is only used to reduce the total number of flow records, and thus, the computational cost.

In the second step, our analyzer searches for trigger-action patterns at each time window. In the monitored network, it looks for suspicious flows with the same destination IP and protocol across all hosts which are presumably receiving commands, and labels them as triggering flows. By studying a collection of contemporary bot samples, we find that bots usually connect to the same group of C&C servers or peers to receive commands and execute the commanded actions immediately upon their receipt from botmasters. For instance, in the case of Storm bot, an infected host locates the botmaster's IP address through its peers from a hard-coded list, receives email lists and templates from the botmaster, and then sends out spam emails within 5 minutes of infection. On the contrary, benign hosts rarely visit the same IP with the same protocol after we filter out the internal and legitimate flows. It is therefore reasonable to associate all of the flows that follow each triggering flow on a host-by-host basis within a time window. These associated flows are considered action flows initiated by triggering flows. Our analyzer then extracts a set of features from each associated flow group to transform it into a flow feature vector for ease of clustering. Note that there is a possibility that benign hosts visit the same IP with the same protocol. Even so, since their flow patterns are usually different, they cannot form clusters among themselves. We will detail this scenario in Section 5.3.

Table 2. Flow features

Index	Flow Features
1 to 4	Duration Mean, Variance, Skewness and Kurtosis
5 to 8	Totalbytes Mean, Variance, Skewness and Kurtosis
9 to 12	Number of Packets Mean, Variance, Skewness and Kurtosis
13	Number of TCP Flows
14	Number of UDP Flows
15	Number of SMTP Flows
16	Number of Unique IPs Contacted
17	Number of Suspicious Ports

Since a flow record is only a brief summary of a session or a connection, the information provided is limited. We make the most of the information by selecting 17 features to constitute a flow feature vector which characterizes not only general traffic patterns but also distinction between benign and malicious hosts at network level. We did so because selecting features essential to all types of botnets can make clustering more effective and accurate, even if our clustering algorithm searches for similarly-behaving hosts and does not require *a priori* knowledge of benign and malicious behaviors. Table 2 shows our selections which are mostly statistical features. Note that features 1 through 14 characterize flow patterns only, which are the sample mean, variance, skewness and kurtosis of flow duration, total bytes transferred, the number of packets transferred, and TCP & UDP break-downs. Features 15 through 17, which are also captured at host level for validation purpose, reveal bots' malicious intent to some degree. For example, spambots usually send a large number of mails using SMTP. Bots also try to reach a large number of unique IPs by scanning or exploiting vulnerabilities at pre-defined ports. Note that benign hosts seldom conduct above activities. Therefore, even if a group of benign hosts visit the same destination themselves or the same as bot-infected hosts do, and cannot be filtered out by the trigger-action association, they may be ruled out by our clustering module because their network behaviors are usually different among themselves and different from bot-infected hosts. Compared to bot-infected hosts, benign hosts are less likely to take similar actions after visiting the same IP because they are not coordinated and commanded to do so.

3.2.2 Clustering

Using a vector representation, each associated group of flows becomes a flow feature vector at every time window; this facilitates the task of clustering. Our goal is to group similarly-behaving hosts together by computing the closeness of their feature vectors. In the area of data clustering, two types of algorithms are available: hierarchical and partitional. We use the hierarchical clustering because its clustering result is deterministic and has a structure that is more informative than the result generated by a partitional algorithm. Using the structured result, we can employ a technique to find a good cut of clustering. Specifically, we use the p-value to gauge the degree each clustering is supported by data. We will detail this clustering in Section 4.2.

3.3 Correlation Engine

As described earlier, whenever a group of hosts is identified by the clustering module as a cluster, the respective host analyzers are required to report the suspicion levels along with network statistics to the correlation engine, since the results generated by flow analysis alone may not be accurate and further in-host validation is needed. Given the two sources of information as input, the correlation engine produces a detection result for each host.

Based on the consistency check of network statistics, there are two possibilities. First, the network features sent from a host may be falsified and thus differ from those observed at the network level. In such a case, the correlation engine considers the host compromised and generates the detection result immediately. Another possibility is that the network-level results are consistent, then we need to consider both the in-host suspicion-level and the quality of the clustering. The detection result should be a function of these two parameters. It is straightforward that the higher the suspicion level the more likely a host is part of a botnet. To quantify the contribution of the clustering quality, we need a measure that reflects the closeness of each host to its clustered group. In other words, the more similar a host's network behavior is to other hosts in the same cluster the more likely it is part of a similarly-behaving botnet. This measure can be the average distance from a specific host to other hosts.

Now, we have two parameters in the correlation algorithm. One is the suspicion level denoted by S_n, and the other is the average distance denoted by $\overline{D_n}$. The final detection score is denoted by $Score_n$ and given by

$$Score_n = w_1 * S_n + w_2 * f(\overline{D_n}). \quad (2)$$

f is a function that maps each average distance $\overline{D_n}$ to a value in [0,1], having the same range as that of S_n. Function f is determined by the distance method being used. w_1 and w_2 are two weight factors. Since at the beginning we cannot completely trust the host-level information, we assign w_1 a lower value and w_2 a higher value, meaning that our detection relies more on the network-level analysis, which is especially important when a host analyzer is compromised. As more consistency check results are obtained, the trust can be built and w_1 increases while w_2 decreases until they become (about) the same.

4 Implementation

We now describe the implementation of each component and the associated overhead of our framework.

4.1 Host Analyzer

Our host analyzer consists of two modules: in-host monitor and in-host suspicion-level generator. As described earlier, in-host monitors capture runtime system-wide behavior at the Registry, file system, and network stack. The implementation of the in-host monitors was adapted from the per-process monitors used in our previous work [24]. Every in-host monitor consists of three sub-monitors. The sub-monitors at the Registry and file system implemented system-call hooking that intercepts related-system calls, stores the passed parameters and status information in a kernel buffer, and then copies them to the user-level application. The two sub-monitors log complete information of every activity of interest, including timestamp, request type and path. The sub-monitor at the network stack was implemented based on WinPcap library and monitors all incoming and outgoing traffic of the host. It collects information including source and destination IPs, ports, and the protocol.

Recall that each host's runtime behavior is transformed to a *behavior vector* for suspicion-level analysis. A behavior vector contains 9 features (Table 1), each of which is represented by a tuple <feature index:value>. For example, the first tuple below means the host created 2 files in the system directory.

0:2 1:2 2:1 3:1 4:2 5:3 6:40 7:55 8:40 [00:10:51, 01:10:51]

The in-host suspicion-level generator employs a SVM algorithm, which produces a value within the interval [0, 1], indicating how likely a host is bot-infected. The SVM needs to be trained by benign and malicious behavior vectors in order to generate a suspicion level for an unlabeled vector. Since bots' in-host behaviors are similar to other types of malware such as network worms, we did not confine our training data to bot-infected hosts but also included other malware-infected ones. Benign hosts' training traces were obtained directly from malware-free hosts which were in normal use. Note that the training data were only used for the SVM to create a classification rule and completely different from the test set in evaluation. Recall that we need an estimated average of suspicion level every time window by using EWMA. The in-host generator produces a suspicion level every 10 minutes and reports the average to the correlation engine on an hourly basis. Thus, $t_n - t_{n-1} = 10$ and W, the total period of time, is set to $1 * 60 = 60$ minutes. S_n is given by:

$$S_n = (1 - e^{-\frac{1}{6}}) * Y_n + e^{-\frac{1}{6}} * S_{n-1}.$$

4.2 Network Analyzer

The network analyzer contains a flow analyzer and a clustering module both of which were implemented in Perl and R. The flow analyzer obtains flow data—an average of 3,900,000 flows per hour—from a core router in our campus network. It filters out internal flows and legitimate flows as described in Section 3.2.1. After the filtering, it removes a majority of flows that are not of interest, reducing to an average of 25,000 flows including 2,000 hosts per hour. The flow analyzer then looks for suspicious flows connecting to the same destination IP and using the same protocol, and labels them as triggering flows. All flows from the same host that follow a triggering flow within the time window are associated with that flow and are suspected to be "action" flows. The flow analyzer finally represents each associated group with a 17-dimensional flow feature vector, the same format as the in-host behavior feature vector. We observed that the final number of hosts to be clustered can be reduced from 2000 to less than 100. We found that bots within the same botnet all connect to the same IPs. Evidently, IRC- and HTTP-based bots talk to their C&C servers. In the hybrid-P2P-based case, Storm instances bootstrap by connecting to the IPs in a hard-coded list, making their contacted IP lists look alike. Waledac instances demonstrate a similar behavior.

The clustering module utilizes hierarchical approach to group flow feature vectors into different clusters. Specifically, we use the *pvclust* package to calculate *p-values* via multi-scale bootstrap resampling for each cluster in hierarchical clustering. The p-value of a cluster is a value in [0, 1], indicating how strong the cluster is supported by data. The package provides two types of p-values: AU (Approximately Unbiased) p-value and BP (Bootstrap Probability) value. AU p-value is computed by multi-scale bootstrap resampling, a better approximation to the unbiased p-value than the BP value computed by normal bootstrap resampling [3]. For a cluster with AU p-value greater than 0.95, the hypothesis that "the cluster does not exist" is rejected with a significant level (equal to or less than 0.05). We thus accept a cluster if its AU p-value is greater than 0.95. The distance measure used in the hierarchical clustering is the "correlation" method. We do not use other distance measures because the correlation values in our data set are mostly positive. A study [7] has shown that in this scenario, the "correlation" method performs best. This method is detailed next.

4.3 Correlation Engine

The correlation engine takes the clustering results and the respective host-based information as input and outputs each host's detection result. Recall that the suspicion level and clustering quality are two parameters in Eq. (2). The clustering quality is represented by the average distance from one host to other hosts in the same cluster calculated by "correlation" method. Assume that a cluster consists of n hosts each of which is represented by a 17-dimensional flow feature vector, forming a $17 \times n$ matrix $X = \{x_{ij}\}$. The i-th row corresponds to the i-th feature of these hosts and the j-th column corresponds to a flow feature vector. The distance between host u and host v is given by

$$D_{uv} = 1 - \frac{\sum_{k=1}^{17}(x_{ku} - \overline{x_u})(x_{kv} - \overline{x_v})}{\sqrt{\sum_{k=1}^{17}(x_{ku} - \overline{x_u})^2}\sqrt{\sum_{k=1}^{17}(x_{kv} - \overline{x_v})^2}}.$$

where $\overline{x}_u = \frac{1}{17}\sum_{k=1}^{17} x_{ku}$ and $\overline{x}_v = \frac{1}{17}\sum_{k=1}^{17} x_{kv}$. As the correlation is always in the range of [-1,1], D_{uv} belongs to [0,2] and so does the average distance $\overline{D_n}$. Recall that the smaller the average distance, the more similarly-behaving a host to other hosts in the cluster and the more likely it is part of a botnet. To reflect this concept in Eq. (2), we selected a decreasing function $f(\overline{D_n}) = 1 - \frac{\overline{D_n}}{2}$ such that the range of f is also [0,1]. Thus

$$Score_n = w_1 * S_n + w_2 * (1 - \frac{\overline{D_n}}{2}).$$

The two weight factors w_1 and w_2 are set to 0.1 and 0.9 at the beginning to reflect the lack of confidence in host-level information. Every time the network feature consistency check passes, w_1 increases by 0.05 and w_2 decreases by 0.05 until they reach 0.5. The final detection $Score_n$ is a value in [0,1].

4.4 Overhead

One may want to know the overhead incurred by the three components of our framework. To measure the overhead of the host analyzer, we used a common Windows benchmark PassMark Software, PerformanceTest [2]. Our host analyzer was implemented on a machine with AMD Athlon 64 3200+ Processor 2.0GHz, 1GB of memory, 80GB of disk, and Windows XP operating system. We ran the benchmark program for CPU, memory and disk, respectively, 5 rounds each. The average overhead for CPU is 3.1%, memory 3.5% and disk 4.7%. The in-host suspicion-level generator can determine one host's suspicion level in about 10 μs given the behavior vector. Since the SVM is pre-trained (i.e., the support vectors are pre-loaded), the training process will not incur any runtime overhead to the host. Our network analyzer and correlation engine were implemented in Linux kernel 2.6.18 on an HP ServerBlade with 2 Dual-Core AMD Opteron(tm) Processors 2.2 GHz, 4 GB of RAM, and 260 GB of disk space. The network analyzer can parse 1-hour flow data and cluster similarly-behaving hosts within 2 minutes on average. To assign the final suspicion score and produce a detection result, the correlation engine spends 1 second per host on average.

5 Evaluation

5.1 Data Collection

We have evaluated the performance of our framework in detecting 3 types of botnets with real-world traces—IRC-based,

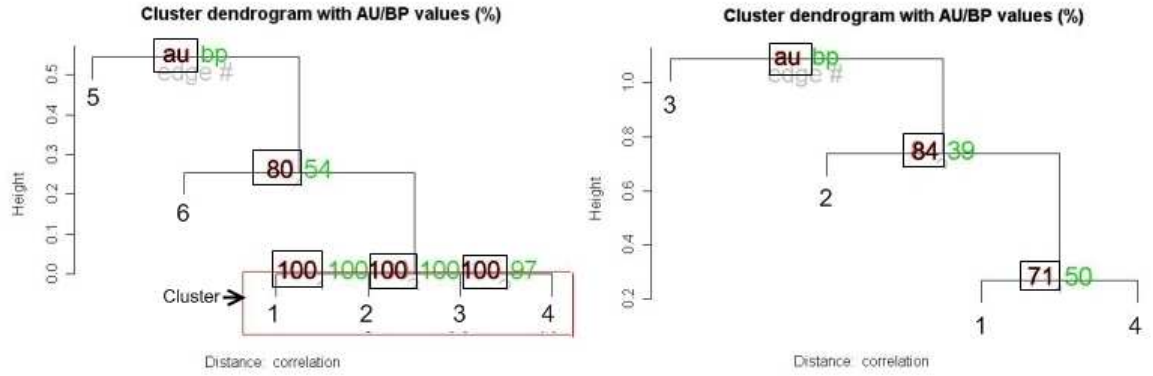

Figure 2. Cluster dendrogram with AU/BP values (%) (Left graph: clustering of bots and benign hosts; Right graph: clustering of benign hosts)

HTTP-based, and hybrid-P2P. We set up VMWare virtual machines running Windows XP, connected via a virtual network to monitor and collect traces. While running these botnets, we also ran a variety of benign applications at the same time to make these machines behave similarly to real compromised hosts. Both the benign and malicious behaviors at the Registry, file system, and network stack were captured. Table 3 shows the details of these botnet traces, each containing 4 bot instances. The modified source code of IRC-rbot and IRC-spybot were used in the virtual network to generate their respective traces. We obtained the binaries of HTTP-based BobaxA and BobaxB, and hybrid-P2P-based Storm and Waledac from public web sites. The IRC- and HTTP-based botnets' network-level traces were captured within a controlled environment and transformed from packet data to flow data in our experiment. Since Storm and Waledac botnets were still active in the wild at the time when we collected data, we carefully configured the firewall setting and connected virtual machines to the external network so that the bots actually joined the real Storm and Waledac botnets, and our campus router captured all of the bots' traffic.

We also collected 5-day NetFlow data from a campus network core router which covered the flows generated by Storm and Waledac instances and all other hosts in the network. Each day's data contained an average of 95,000,000 flows. We were not able to capture host-level data on every single machine in the campus network except those under our control. But our campus network administrator confirmed that all hosts in the 5-day flow data except for those running Storm and Waledac were benign, meaning that there was no botnet traffic present in other hosts during that period. Thus, it is valid to assume that in our evaluation, these hosts are benign at both host- and network-level (other types of malware might run on those hosts but they are not our detection targets). The 5-day data consists of three sets: (1) 2-day data that contains 48-hour Storm traces; (2) 1-day data including Waledac; and (3) other 2-day data. We divided the third data into two subsets, 1-day each. Note that overlaying malicious traffic on clean traffic for evaluation has commonly been used in malware detection literature [12, 11, 23]. Although our botnet traces already contained benign traffic, the amount of such traffic was limited and we wanted to add more to make it more realistic . Thus, we overlaid the botnet network traces except Storm and Waledac, one at a time, on data set (3), two traces on the first day, and two on the second day. For example, the IRC-rbot included 4 bot instances, so we randomly selected 4 hosts from the clean 1-day traffic and replaced the bots' IPs with the selected IPs. We treated Storm traces in the same way and intentionally overlaid the 1-day Waledac traffic on HTTP-intensive benign hosts, the purpose of which will be described later. In addition, hosts running P2P clients are important for the evaluation of our detection framework as one may wonder if they will be misclassified as bots. Since NetFlow data could not reliably tell which host had such applicatioin, we ran P2P applications such as eMule and BitTorrent on hosts under our control and collected their host- and network-level traces. Specifically, we obtained 4 sets of hour-long traces for hosts running eMule and 3 sets for those running BitTorrent. While conducting P2P activities, those hosts also ran other regular network-relevant applications, such as web-browsing, ssh and email-checking, similar to what benign hosts usually do. In what follows, we will first report the detection accuracy of our combined approach, and then show the benefit of combining both host- and network-level information.

Table 3. Botnet traces

Trace	Duration	Number of Bots
IRC-rbot	24h	4
IRC-spybot	32m	4
HTTP-BobaxA	4h	4
HTTP-BobaxB	20h	4
Storm	48h	4
Waledac	24h	4

5.2 Detection Results

We now report the detection results on 6 botnets. The performance of our detection framework was measured by false-

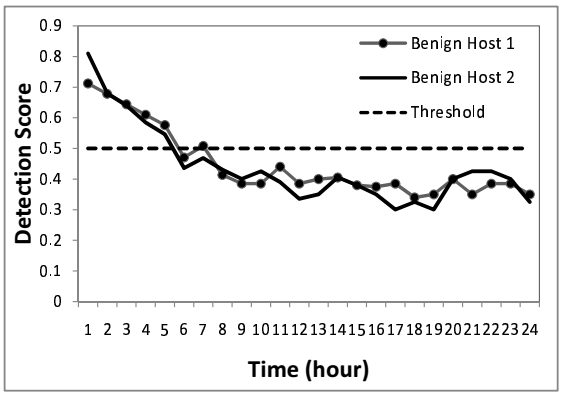

Figure 3. The change of detection scores for two benign hosts

Table 4. False alarm rates

Trace	Avg FP hosts	Avg FP	Avg FN Hosts	Avg FN	Duration
IRC-rbot	3.208	0.0016	0.125	0.0313	24h
IRC-spybot	2.833	0.0014	0	0	24h
HTTP-BobaxA	1.000	0.0005	0	0	24h
HTTP-BobaxB	1.083	0.0005	0	0	24h
Storm	2.563	0.0013	0	0	48h
Waledac	0.9167	0.0005	0	0	24h

alarm rates, i.e., false-positive (FP) and false-negative (FN) rates. A false-positive is defined as a benign host mistakenly classified as bot-infected, and a false-negative means that an actual bot-infected host fails to be detected.

Recall that the detection score is in the interval [0,1]. The detection threshold was set to 0.5 in our evaluation to strike a balance between FP and FN rates, and this parameter is configurable. Note that there is always a tradeoff between FP and FN rates. A lower threshold can be set if FNs are a concern, while a higher threshold may be required if FPs are less desirable.

Table 4 shows our evaluation results where the average number of FP or FN hosts is calculated during the entire period of evaluation. The average FP or FN rate is the number of FP hosts divided by the total number of benign hosts (which was around 2,000), or the number of FN hosts divided by the total number of bot-infected hosts. Our framework was found to be able to identify almost all bot-infected hosts. There was only one bot undetected, generating a false-negative. We verified that this bot did not have any C&C or malicious network activity and thus failed to form a suspicious cluster with other bots.

Our framework also performs well in terms of false-positives. The highest false-positive rate was no greater than 0.16%. It turned out that almost all false-positive hosts appeared during the first few hours of the traces due to the values of "untuned" weight factors w_1 and w_2. As mentioned in Section 4.3, we set w_1 (the weight of suspicion level) to 0.1 and w_2 (the weight of clustering quality) to 0.9 at the beginning to reflect lack of confidence in the host-level information. During the first few hours our framework relied more on the network-level analysis, resulting in detection inaccuracy when a group of benign hosts demonstrated similar network behaviors among themselves (e.g. they ran the same network applications) or behaved similarly to bot-infected hosts. As the host-level information was verified to be trustable, w_1 was increased and w_2 was decreased so that host-level information gradually had a higher weight and was able to correct the detection results. Figure 3 shows the change of the detection scores on two benign hosts which have similar network traffic patterns and form a cluster by themselves. At the beginning, both of them have greater than 0.5 detection scores due to the high weight assigned to the clustering quality parameter, leading to false-positives. As time goes by, a host's suspicion-level parameter receives a more balanced weight. Since their suspicion levels are always low (0 to 0.1), their final detection scores decrease below the 0.5 threshold and no longer incur false-positives. Admittedly, since we were not able to obtain host-level data from all machines in the network, the false-positive rate might be underestimated. But we have tried our best to run a variety of benign applications on the hosts under our control and used their traces in our evaluation. They did not cause many false-positive alarms as shown before. We believe that network- and host-level information complement each other, and hence combining them while making a detection decision is the key to reducing false alarm rates.

5.3 Evaluation with Network Analyzer

Using the network analyzer that performs flow analysis and clustering, we found some interesting results. As pointed out in Section 4.2, the trigger-action association done by the flow analyzer can significantly narrow the number of hosts for clustering because benign hosts rarely visit the same IP with the same protocol after traffic filtering, while bot-infected hosts connect to the same group of C&C servers or peers. Even if benign hosts cannot be filtered out by trigger-action association, since they visit the same destination or behave the same as bot-infected hosts do, they are likely to be discarded by the clustering module, i.e., their flow patterns are usually different among themselves and different from bot-infected hosts. This fact makes the clustering module effective in reducing the number of benign hosts appearing in the final clusters.

Figure 2 shows the hierarchical clustering dendrogram of scenarios in which a few benign hosts were ruled out not by the trigger-action association but by the clustering module. The graph on the left is the scenario when bot-infected and benign hosts happened to visit the same destination and their flow feature vectors were sent to the clustering module for grouping. There are 6 hosts to be clustered, numbered from 1 to 6. 1 to 4 are bot-infected hosts, and 5 to 6 are benign hosts. Recall that we use hierarchical clustering with AU p-values indicating how strong the clustering is supported by data. Normally, clusters with p-values greater than 95% are considered reasonable clusters. The AU p-values and reasonable clusters are highlighted by rectangles in the figure. In the left graph, 4 bot-infected hosts are clustered together with 100% AU values, meaning that their flow feature vectors are quite similar.

The two benign hosts in the graph cannot form a cluster with them because of the dissimilarity in flow patterns between the benign and bot-infected hosts. The graph on the right represents the scenario when a few benign hosts visited the same destination. 4 hosts, numbered from 1 to 4, are all benign. The 4 benign hosts cannot make any cluster among themselves (low AU p-values), since their flow feature vectors differ significantly and cannot be grouped together.

We also collected additional data from several benign hosts running P2P applications to see if they would generate false-positives. Four hosts each ran an eMule client, and three other hosts each ran a BitTorrent client named *utorrent*. Besides P2P file sharing, these hosts also made other network accesses during the period of trace collection including web-browsing, ssh, and email-checking. This is realistic because normally a P2P user does multi-tasking during file-sharing, rather than solely waiting for the file-sharing to complete. We used the network analyzer to perform flow analysis and clustering on these P2P data sets. It turned out that the four eMule hosts did visit the same IPs (servers) so that they were not ruled out by the trigger-action association and needed to be clustered. The same thing happened to the three utorrent clients. However, during the clustering, those P2P hosts could not make any cluster. We found that the AU p-values generated for the four eMule hosts were no greater than 85% and for the three utorrent ones no greater than 90%, both of which were below the 95% bar. That is, these benign hosts did not behave similarly at the network level even though they ran the same P2P client. One reason for this is that P2P file-sharing is a user-specific activity. Users have different interests and download or upload different files so that the flow features, such as total bytes, number of packets and number of TCP or UDP flows are hardly similar. The other reason is that network activities other than P2P also add some dissimilarity to the flow patterns among hosts running P2P applications. Although in our experiment, P2P hosts were ruled out by the clustering module, we still inspected their host-level behaviors to make sure that even if the network analyzer failed to distinguish them, the host analyzer could tell they were benign. These results were in line with our expectation: the suspicion-levels for those hosts were always much less than 0.5, because there was little malicious behavior demonstrated at the host level. In this scenario, since the correlation engine considers both types of information, it will generate correct detection results with the help of suspicion-levels.

In summary, our network analyzer—the flow analyzer along with the clustering module—is effective in forming suspicious clusters. But it may fail in some situations where the host analyzer needs to come in and help. We described above a hypothetical scenario for the benign P2P case, but will present a real case study next.

5.4 Evaluation with Host Analyzer: A Case Study of Waledac

Waledac worm spreads as an attachment to a spam email or through a link to a malicious website. It came into the wild at the end of 2008 and still remains active (as of Dec 2009). After infecting a host, Waledac searches for email addresses from the host and sends the information to a set of hard-coded IP addresses. It uses the HTTP protocol for C&C traffic forwarding and the botmasters are well hidden behind a P2P network [9]. We downloaded samples by following Waledac spam's links to its malicious domains in Feb 2009.

As mentioned earlier, we ran Waledac instances on virtual machines, and they joined the actual Waledac botnet for 24 hours. We intentionally overlaid their network traces on benign hosts with heavy HTTP traffic. We did this because Waledac appeared stealthy in its network activities, and we wanted to evaluate how well our framework can perform in the situations where the network analyzer cannot distinguish between benign and bot-infected hosts. We observed that these Waledac instances did not send any spam emails in the 24-hour period. The only activity was several HTTP sessions every hour for C&C such as transferring locally-collected information. This type of malicious traffic is easy to blend into benign HTTP flows but hard to isolate.

Over a few time windows, our network analyzer mistakenly clustered one benign host into the same group as 4 Waledac bots. One reason lies in the way we mixed benign and bot's traffic for bot-infected hosts (we did this intentionally). It turned out that the HTTP-intensive benign host and the 4 bot-infected hosts had visited the same destination IP using the HTTP protocol. As shown before, only visiting the same IP is not enough for forming a cluster. To be grouped into the same cluster, the hosts should have similar traffic patterns. For most bot-infected hosts, although their flows are mixed with benign flows, their malicious flow patterns may still be conspicuous because of their distinct and aggressive spam sending and scanning activities. In other words, their network-level behaviors can be distinguished from benign hosts' during the clustering. However, in the Waledac's case, the stealthy C&C traffic was hidden and diluted into the benign HTTP traffic, and the clustering module failed to differentiate bot-infected hosts' network activities from the benign hosts', which is the other reason for the incorrect clustering. Nevertheless, our framework still correctly generated the detection results for Waledac bot instances, thanks to the information obtained by the host analyzer. As the Waledac exhibited malicious behavior at the host level, such as creating an autorun Registry key, and dropping or modifying files in the system directory, each bot-infected host's suspicion level was around 0.88 on average. On the other hand, the benign host's suspicion level was close to 0. The final detection $Score_n$ for bot-infected hosts was as high as 0.85, while that for benign hosts was 0.40, showing a significant difference. Without the host analyzer, benign hosts are likely to be misclassified as bot-infected ones in the pres-

ence of bots that are stealthy at the network level. In other words, relying solely on a network-level analysis cannot create a complete picture: the inspection of in-host behavior by the host analyzer is critical in reducing false-positives.

6 Discussions

In this section, we discuss several limitations of our framework and their potential solutions.

One possible limitation of our approach is its scalability since the approach requires runtime host-level analyzers. Our design is intended for use in enterprise networks where a security framework can be enforced easily on all hosts. In large-scale networks, if host analyzers cannot be installed on every host, we may use available host analyzers to infer the suspicion levels at those without the analyzers if they form the same suspicious cluster by network-level analysis. Delving more into this issue is part of our future work.

Since our network analyzer looks for trigger-action patterns among hosts, bots may delay their coordinated actions by waiting for random period of time. To counter this evasion attempt, we may lengthen our time window of analysis, such as a day as opposed to a few minutes or hours. We can also randomly select a time window that cannot be figured out by the attackers. As the goal of a botnet is to perform malicious actions, if each bot does not act maliciously for a very long time, it will be ineffective, causing few problems. Bots may also attempt to randomize their traffic patterns such as injecting random number of packets in each flow or they can mimic flow patterns of benign hosts. However, these techniques may not help bots much to evade our detection because our framework also considers host-level behavior while making detection decisions.

Since our framework is deployed in a monitored network, hosts within the network are geographically close to one another. It is convenient for bots to connect to, or bootstrap from, the same set of nearby IPs to receive commands and take actions in a coordinated manner. To intentionally evade our network analyzer, bots may use different C&C servers or contact a different set of IPs. If there are a large number of bots in our network, our approach may group them into several suspicious clusters. However, if there is only one or a few bots (without contacting the same set of IPs), our detection framework should obtain information from the host analyzers more frequently. In that case, host analyzers can send the suspicion-level information to the correlation engine before any suspicious clusters are formed. If the suspicion level is high enough, the detection result can be generated even without any suspicious cluster.

Another possible evasion of our framework is to compromise the host analyzers and send falsified information to the correlation engine. This can happen only if the bot sits below our host monitoring level and is able to modify or subvert the system-wide information the in-host monitor receives. Our current solution is to gather a few network statistics from the host to compare against those observed by the network analyzer. It is possible that a bot keeps the network statistics intact and modifies the suspicion-level information only to mislead the correlation engine. Even if this happens, there is still a high possibility of capturing the bot, because, as described in Section 4.3, the weight factor assigned to the host-level information by the correlation engine is much lower than the weight factor assigned to the network-level information (i.e., clustering quality) in the beginning. It means that with a high weight factor on network information, as long as the compromised host exhibits correlated malicious network-level activities and forms a suspicious cluster with other hosts, it will be detected with high probability despite the falsified suspicion-level sent from the host analyzer. To further counter this attack, we may also use secure hardware or secure VMM to safeguard the OS as well as our monitors in each host.

While there is a possibility that our framework could be evaded, the evaluation results demonstrated that our system performed well in detecting state-of-the-art botnets with minimal impact on benign hosts. Therefore, our system raises the bar against botnets.

7 Related Work

Numerous approaches have been proposed for detection of botnets. Most of them target centralized botnets, i.e., IRC-based and HTTP-based. Gu *et al.* [12] used network-based anomaly detection to identify centralized botnet C&C channels based on their spatial-temporal correlation. Binkley *et al.* [5] combined an IRC mesh detection component with a TCP scan detection heuristic. Rishi [10] is a botnet detection system that relies on IRC nickname matching. Karasaridis *et al.* [14] proposed detection of botnet controllers by flow aggregation and feature analysis. Livadas *et al.* [17, 22] utilized supervised machine learning to classify network packets in order to identify the C&C traffic of IRC-based botnets. As hybrid P2P botnets emerged, some researchers studied the Storm botnet and proposed approaches tailored to P2P-based botnet detection. Holz *et al.* [13] measured the size of the current Storm net by infiltrating through a crawler, and proposed mitigation strategies that introduce controlled peers to join the network to either separate or pollute the content of the Storm network. Porras *et al.* [19] tried to detect the Storm bot by constructing Storm's dialogue lifecycle model first and then identifying the traffic that matches this model, a notion similar to that in [12]. Compared to the above approaches, our framework is a general detection scheme that is not constrained by any C&C protocol and does not require the learning of C&C profiles prior to detection.

To the best of our knowledge, only BotMiner [11] is designed for protocol-and structure-independent botnet detection. It clusters similar communication and malicious traffic, and performs cross-plane correlation to identify the hosts that share both patterns. TAMD [23] aimed to detect infected hosts within a network by finding those that share common and unusual network communications. It employs external destination, payload and OS platform aggregation functions to group hosts. TAMD and BotMiner rely solely on network-level anal-

ysis, whereas our framework combines both network- and host-level information to complete a detection picture. Moreover, both TAMD and BotMiner require packet-level inspection in the traffic analysis, while our network analyzer only looks into Netflow data avoiding privacy issues and large computational costs. Another difference worth noting is that our network analyzer looks for trigger-action patterns before forming clusters, which was not used by previous work. This technique significantly reduces the number of benign hosts for clustering, making the network-level analysis more efficient.

As for host-based solutions, there are many general detection approaches [6, 15, 8, 21, 20]. They either use signature matching or behavior analysis by system or API call sequence modeling. Unless a virtual machine monitor is integrated into those techniques, they can be disabled or compromised if there is malware sitting below the detection framework. Our approach does not rely on one single source of information; it incorporates both network- and host-level behavior. Also, our host analyzer sends a few network metrics to the correlation engine for validation, which adds an additional layer of security.

8 Conclusion

Considering the coordination of bots within a botnet and each bot's malicious behavior at the host level, we proposed a C&C protocol-independent botnet detection framework that combines both host- and network-level information. Our network flow analyzer searches for trigger-action traffic patterns among different hosts without accessing the packets' payloads, and clusters similarly-behaving hosts into suspicious groups. Our host analyzer then obtains suspicion-level information along with a few network statistics on a host-by-host basis for verification. Finally, our correlation engine generates a detection result for each host by taking into account both suspicion-level and clustering results. Our experimental evaluation based on real-world data has shown the following results. The network analyzer can be effective in forming suspicious clusters of aggressive bots but may fail to separate benign hosts from bot-infected hosts if the latter are stealthy at the network level. When the stealthy bots are present, it is the host analyzer that provides correct detection results by generating distinguishing suspicion levels. By using combined host- and network-level information, our framework is shown to be able to detect different types of botnets with low false-positive and false-negative rates.

References

[1] Netflow. http://www.manageengine.com/products/netflow/cisco-netflow.html.

[2] Passmark. http://www.passmark.com/.

[3] The pvclust package. http://www.is.titech.ac.jp/ shimo/prog/pvclust/.

[4] Storm worm. http://en.wikipedia.org/wiki/Storm-Worm.

[5] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, Berkeley, CA, USA, 2006. USENIX Association.

[6] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantics-aware malware detection. In *Proceedings of IEEE Symposium on Security and Privacy*, 2005.

[7] E.F.Glynn. Correlation "distances" and hierarchical clustering. http://research.stowers-institute.org/efg/R/Visualization/cor-cluster/index.htm, Dec 2005.

[8] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. *IEEE Symposium on Security and Privacy*, 1996.

[9] S. Foundation. Waledac is storm is waledac? peer-to-peer over http.. http2p? http://www.shadowserver.org/wiki/pmwiki.php/Calendar/20081231, Dec 2008.

[10] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, Berkeley, CA, USA, 2007. USENIX Association.

[11] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[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] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *In Proc. First USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, 2008.

[14] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *HotBots'07*, Berkeley, CA, USA, 2007. USENIX Association.

[15] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based spyware detection. In *Proceedings of the 15th USENIX Security Symposium*, 2006.

[16] H. Lin, C. Lin, and R. Weng. A note on platt's probabilistic outputs for support vector machines, 2003.

[17] C. Livadas, R. Walsh, D. Lapsley, and W. Strayer. Using machine learning technliques to identify botnet traffic. In *Proceedings of the 2nd IEEE LCN Workshop*, Nov, 2006.

[18] J. Nazario. Walking waledac. http://asert.arbornetworks.com/2009/01/walking-waledec/, January 2009.

[19] P. Porras, H. Saidi, and V. Yegneswaran. A multi-perspective analysis of the storm(peacomm)worm. Technical report, SRI, 2007.

[20] R. Sekar, M. Bendre, P. Bollineni, and D. Dhurjati. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2001.

[21] A. Somayaji and S. Forrest. Automated response using system-call delays. In *Proceedings of the USENIX Security*, 2000.

[22] W. T. Strayer, R.Walsh, C. Livadas, and D. Lapsley. Detecting botnets with tight command and control. In *Proceedings of the 31st IEEE LCN*, November, 2006.

[23] T. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *Proceedings of the 5th GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2008.

[24] Y. Zeng, X. Hu, A. Bose, H. Wang, and K. G. Shin. Containment of network worms via per-process rate-limiting. In *Proceedings of 4th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2008.